Code Ownership: Impact on Maintainability

Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc

University of Szeged, Department of Software Engineering
Dugonics tér 13, H-6720 Szeged, Hungary
{farago,hpeter,ferenc}@inf.u-szeged.hu

Abstract. Software systems erode during development, which results in high maintenance costs in the long term. Is it possible to narrow down where exactly this erosion happens? Can we infer the future erosion based on past code changes?
In this paper we investigate code ownership and show that a further step of code quality decrease is more likely to happen due to the changes in source files modified by several developers in the past, compared to files with clear ownership. We estimate the level of code ownership and maintainability changes for every commit of three open-source and one proprietary software systems. With the help of Wilcoxon rank test we compare the ownership values of the files in commits resulting maintainability increase with those of decreasing the maintainability. Three tests out of the four gave strong results and the fourth one did not contradict them either. The conclusion of this study is a generalization of the already known fact that common code is more error-prone than those of developed by fewer developers.
This result could be utilized in identifying the "hot spots" of the source code from maintainability point of view. A possible IDE plug-in, which indicates the risk of decreasing the maintainability of the source code, could help the architect and warn the developers.

Keywords: Code ownership, ISO/IEC 25010, source code maintainability, Wilcoxon test

1 Introduction

Software quality plays a crucial role in modern development projects. Maintainability is one of the six sub-characteristics of software quality, as defined originally in the ISO/IEC 9126 standard [14]. Software maintenance consumes huge efforts: based on the experiences, at least half 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 quality, which could either help to avoid software erosion, or provide information about how to better allocate efforts spent on improving software quality.

In a recent paper [9], 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 [8] that revealed the connection of the version control operations and maintainability. It turned out that file additions in a project have rather positive, file updates (i.e. propagation of the changes in the existing source code) have rather negative effect on maintainability, while a clear effect of file deletions was not identified. Furthermore, in a more recent work [7] we presented the results of a variance analysis. File additions and file deletions increase the variance of the maintainability, and the update operation decreases it.

In this work we make use of the author information coming from the version control system and check the effect of code ownership on maintainability. We performed the analysis on commit basis. We collected historical data form SVN version control system and estimated the maintainability with the help of ColumbusQM probabilistic software quality model [1]. We defined the code ownership values based on the historical changes.

Formally, we investigated the following research question:

Research Question: *Does the number of developers modifying the same code in the past have any affect on the maintainability change of future commits?*

Null hypothesis: *The past does not have any influence on the future: the future maintainability time line of the source code is totally independent of the number of developers modifying a code in the past.*

Assumed alternative hypothesis: *Modifying files without clear ownership (i.e. those of which have been modified by several different developers in the past) is more likely to result in further maintainability decrease than modifying files with clear ownership (i.e. those modified by only one or by a very few number of developers).*

We investigated this question by studying three open-source systems and an industrial one. According to the results presented later in this paper, we rejected the null hypothesis. The performed Wilcoxon rank test varied in significance (p-values of 0.000014, 0.034, 0.060 and 0.21). The only not significant result was caused by a software system where more than 80% of the total commits were performed by a single developer, therefore we can consider that as an outlier from this respect. This supports our assumption that modifying common code (i.e. source files which have been modified by several developers in the past, according to the source control log) is more likely to cause software degradation and decrease the maintainability compared to those modifications affecting a code with clear ownership.

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 – how we collected the data, what kinds of tests were performed and how we illustrated the results. In Section 4.1 we provide details about the software systems we used for our study. In Section 4 we present the results of the statistical tests. In Section 5 we list the possible threats to the validity of the results, while we conclude the paper in Section 6.

2 Related Work

There are several papers dealing with the topic of code ownership or developer related issues.

In their work Mockus et al. [16] present a case study of the Apache Server open source development. Among others they considered the topic of code ownership as well. They analyzed a single project which had nearly 400 contributors and concluded that in the analyzed project no real code ownership was evolved. We analyzed 4 systems, with the magnitude of 1-3 dozens of developers each and analyzed the effects of the code ownership on future maintainability.

In a study, Nordberg [17] describe four types of code ownership: product specialist, subsystem ownership, chief architect and collective ownership. They discusses the advantages and disadvantages of each models. Our findings support the base assumption of this study: in case of lack of well defined code ownership the code quality is likely to decrease. We did not consider code ownership models in such detail, but present the most obvious developer related facts for the 4 analyzed systems.

LaToza et al. present the results of two surveys and eleven interviews [15] conducted by software developers at Microsoft, regarding software development questions. Some of the questions were related to code ownership as well. An interesting statement of this article is that code ownership can also be wrong, as if a code is understood and maintained by a single developer, it makes individuals too indispensable. As an alternative of individual code ownership, the team code ownership was also investigated. Contrary to them, we examined the effect of the code ownership on the maintainability, i.e. studied why code ownership is good, but from the organizational level the aspects can be different in the longer term.

Fritz et al. investigated the frequency and recency of interactions [10] on the code by developers: questions were asked to find out if they can recall types of variables, types of parameters, method names, another method calls and methods which calls a specified method. They showed that according to the assumed hypothesis, the developers know their own code better (that was modified by him/her frequently and recently) compared to foreign code. We, on the other hand, analyze code ownership instead of code knowledge.

Weyuker et al. investigate if their already presented fault prediction model can be enhanced by including the number of developers [20]. They found that the achieved improvement is negligible, which might be surprising at a first glance. We, on the other hand, found a significant correlation by examining the number of different developers' effect on maintainability. The contradiction could be resolved by the following: an already well established model cannot be enhanced further significantly by including the number of developers predictor; but it itself is a good predictor of maintainability change and of defects as well.

In their study Bird et al. [3] investigate if there are significant differences in software quality following a distributed development model compared to a collocated development. They analyzed the development of Windows Vista and argue that the differences are hardly notable. As a complementary result they found a positive correlation between the number of developers and defects, which result is similar to ours. In our current work, we did not consider the distance among development team members, but analyze the effect of ownership on software maintainability.

The same authors present a fault prediction method [4], which combines social factors in development organizations and program dependency information. They found that this was a better model than considering only one of the factors. They proved their concept on 2 huge projects: Windows Vista and Eclipse. We also used both social and technical networks implicitly: the social one is the number of developers of a module, and the technical one is the sources committed together.

The problem of code ownership, especially finding the hidden co-authors, is analyzed by Hattori et al. [12]. They created a tool called Syde, which records every change by every commit, and with the help of this information they were able to determine the code ownership more precisely. They validated their concept using a commercial system. We also analyzed code ownership, bud did not consider developer interaction information.

Rahman et al. [19] introduced a code ownership and experience based defect prediction model, but instead of just considering the modifications performed on source file itself, they introduced a fine-grained level by analyzing the contributions to code fragments. We on the other hand performed our analysis conventionally on source file basis.

The study by Bird et al. [5] targets similar goal to ours; as its title says: the effects of ownership on software quality. The authors investigated 2 huge projects: Windows Vista and Windows 7. We, on the other hand, investigated 4 smaller projects. They considered software quality in terms of pre-release faults and post-release failures; we consider code maintainability as an aggregated value of complexity metrics. They performed the analysis on binary and release level; our study is based on source code and commits. For a binary they defined the terms minor contributor (developers who contributed at most 5% of the total commits), major contributor (above 5%) and ownership (proportion of the commits of the highest contributor). Among others, they found that software components with many minor contributors had more failures than other software components. Moreover, the high level of ownership resulted in less defects. These findings are very similar to ours: by increasing the number of developers and therefore decreasing the ownership the software quality tends to reduce.

3 Methodology

3.1 Overview

In order to be able to analyze the connection between code ownership and maintainability, we need a method to express them numerically. Neither of them are trivial concepts, and there are no exact definitions on how to compute them.

For the maintainability we used the same calculation method that we applied in our previous studies. Therefore these values were available even before preparing the current work. The maintainability estimation method is described in detail in Section 3.3. As the used quality model works on a revision basis (it analyzes a certain revision of a system), we had to work on a per commit basis.

One of the most important tasks of this research was to find a proper way to numerically express the code ownership of a commit. The details on how we did this is described in Section 3.4. It was a constraint to deal with code ownership on a commit basis because working with files would have been more natural. Nonetheless, we are convinced that this loss of precision might only weakened the results, and our main conclusions would remain valid using file level ownership values.

In Section 3.5 we argue that the two number series (maintainability and ownership values) are totally independent. Section 3.6 describes the statistical tests we used to verify our hypothesis.

3.2 Preliminary Steps

Before collecting the required information we did some data cleaning. The analyzed software systems were all written in Java and the quality model we used (see below) considers Java source files only. Therefore we removed the data related to the non Java source files (e.g. xml files) from the input. We also removed the commits that became empty (i.e. which contained no Java source files at all). In this way we worked on an input commit set containing exclusively Java source files, and each analyzed revision contained at least one affected Java file.

3.3 Estimation of the Maintainability Change

We estimated the maintainability value of every revision with help of the ColumbusQM probabilistic software quality model [1]. This model is among others based on the fact that the increase of software metrics (e.g. object-oriented metrics defined by Chidamber and Kemerer [6]) decreases the maintainability. Gyimóthy et al. [11] empirically validated that the increase of some of these metrics increase the probability of faults.

The model itself considers the following metrics: logical lines of codes, the number of ancestors, the maximum nesting level, the coupling between object classes, clone coverage, number of parameters, McCabe's cyclomatic complexity, number of incoming invocations, number of outgoing invocations, and number of coding rule violations. These metrics are compared with those of other systems in a benchmark, and then the results of the comparisons are aggregated by utilizing also weights provided by developers.

From this study's viewpoint we treat this quality model as a black box. Details were published in the works of Bakota et al. [1], and they also showed that there is a correlation between the estimated quality value and the real development costs [1, 2]. For this level of abstraction it is enough to know that if all of the metrics increase, then the maintainability decreases; if they all decrease, then the maintainability increases; and if some of them increase and others decrease, then the direction of the maintainability change depends on the benchmark and the aggregation. As a result, we get a sign (positive, zero, or negative) for each commit.

3.4 Code Ownership Calculation

We used the following method to express the code ownership numerically. In a particular commit, we considered all the affected source files one by one. As indicated in Section 3.2, there is at least one Java file in every analyzed commits. For every source file, we calculated how many different developers commited on that file at least once from the beginning of the available history, including the actual commit as well. Therefore this value will be at least 1.

At this point we have a positive integer number for every affected source files of the commit in question. But for further analysis we need a value describing the ownership of the actual commit. For this we chose to calculate the geometric mean of the collected values for files. This expresses well the overall ownership of the file based actual ownership values.

For example, consider a small artificial project with 4 java files: `A.java`, `B.java`, `C.java` and `D.java`. This project have been developed by the following developers: `sulley`, `mike`, `randall` and `celia`. In Table 1 the rows represent commits. The first column contains the revision number, while the second one contains the author of that commit. Then the odd columns indicate if the actual file was affected by the commit in question, and the even columns contain the number of different developers of that file up to the current commit. The last column contains the calculated ownership value.

Table 1. A simple example

Rev.	**Author**	`A`		`B`		`C`		`D`		**Own.**
1	`sulley`	+	1	+	1					1.00
2	`mike`	+	2			+	1			1.41
3	`sulley`	+	2	+	1					1.41
4	`randall`							+	1	1.00
5	`celia`	+	3					+	2	2.45
6	`randall`	+	4					+	2	2.83
7	`sulley`	+	4	+	1					2.00
8	`mike`	+	4							4.00

In the first revision `sulley` added files `A.java` and `B.java`. The ownership values are initialized to 1 for both of the files, and the geometric mean of 1 and 1 is 1.

In the second revision `mike` modified file `A.java` and added file `C.java`. At this point the ownership value of source file `A.java` has been increased to 2 (`sulley` and `mike`), and the value of file `B.java` was initiated to 1. The ownership value of the second revision is $\sqrt{1 \cdot 2} \approx 1.41$.

With these 2 examples the other 6 revisions are easy to understand.

From this scenario the following can be seen:

- File `A.java` is the "hot area" of the "project", modified by every developer.
- File `B.java` is an example of intensive modification by a certain developer (`sulley` in this case), therefore having a clear ownership.
- File `C.java` is an example of adding a source file once and never modifying later.
- File `D.java` is an example of a common code of two developers (`randall` and `celia` in this case).

As a result of the above described method, we get an ownership value for every revision.

3.5 Independence of the Values

At this point we have a maintainability change sign and an ownership value for each commit. Before going further to the statistical tests performed on these data, we argue that the calculated values are totally independent.

On one hand, the maintainability value of the system is calculated solely from the source code, no version control data is considered. The sign of the maintainability change of the actual commit depends (mainly) on the code delta, neither the code history nor the author are taken into account.

On the other hand, the ownership value is calculated solely from version control historical data, particularly the author of the past and the present commits is considered. Therefore we can state that the data series are independent form each other.

3.6 Comparison Tests

We divided the commits into 3 subsets based on the sign of maintainability changes, and analyzed their calculated ownership values. We omitted the neutral maintainability changes (i.e. no change in maintainability values), therefore 2 set of numbers remained:

– ownership values of the commits with positive maintainability change, i.e. code quality increase, and
– ownership values of the commits with negative maintainability change, i.e. code quality decrease.

The null hypothesis is that there is no significant difference between these values. The alternative hypothesis is that the ownership values related to commits with positive maintainability changes are significantly lower than those related to negative maintainability changes.

Considering the limitations of the data (e.g. it is not normally distributed) we chose the Wilcoxon rank correlation test (also known as Mann-Whitney U test) for comparison. This test compares all the elements of the first data set with all the elements of the other one, taking all the possible combinations into consideration. The null hypothesis is that the number of "greater" elements is the same as the number of "less" elements. The alternative hypothesis is that the elements of one of the sets are significantly higher than the elements of the other.

By default the Wilcoxon rank correlation test performs a two-tailed analysis. This means that it tells if the values in the checked subsets differ significantly, but does not tell the direction of the deflection. In this case we were not satisfied with the information if the elements of one subset significantly differ from the elements of the other one, we were also interested in which values were higher. For this reason we executed the one-tailed Wilcoxon-rank test with the alternative of "less". Practically it means that our alternative hypothesis is that in case of the above comparison the number of "less" elements are significantly higher than the number of "greater" elements.

The most important result of this test is the well-known p-value, indicating the probability of the result being at least as extreme as the observed, provided that the null-hypothesis is true. In the results section we present these p-values for the analyzed systems.

We interpret the p-values as follows:

– below 0.01: very strong significance,
– between 0.01 and 0.05: strong significance,
– between 0.05 and 0.1: significant,
– between 0.1 and 0.5: not significant,
– between 0.5 and 0.9: contradiction, and
– above 0.9: the opposite statement is true.

We performed the test using the `wilcox.test()` function in R [18]. R runs on multiple platforms; we performed the evaluation on Windows 7. As a result, we get p-values for all software systems the test was performed on.

Going back to our running example, Table 2 shows how the maintainability changed in each revision. For a better overview we repeated the ownership values in this table as well.

Table 2. Example Maintainability Changes

Revision	**Ownership**	**Maintainability change**
1	1.00	positive
2	1.41	neutral
3	1.41	negative
4	1.00	positive
5	2.45	negative
6	2.83	positive
7	2.00	neutral
8	4.00	negative

Now we have the following ownership value sets:

– Ownership values related to positive maintainability changes: $\{1.00, 1.00, 2.83\}$
– Ownership values related to negative maintainability changes: $\{1.41, 2.45, 4.00\}$

Considering all the comparison combinations (there are $3 * 3 = 9$ cases) we get the following. In 7 cases (the two 1.00 in all comparisons and comparing 2.83 with 4.00) the elements in the first data set are less than the elements in the second one, and in 2 cases (comparing 2.83 from the first data set with 1.41 and 2.45 from the second one) the result of the comparison is just the opposite. The p-value in this example is about 0.19, indicating that the elements in the first subset is less, but not significantly, than the elements in the second set. The obvious reason for this is the small number of observation.

4 Results

4.1 Analyzed Systems

We executed the tests on four software systems. These are four independent systems, i.e. we executed four independent tests. The initial selection criteria for the subject systems were the following: availability of at least 1,000 commit and at least 200% code increase during the analyzed period. The analysis was – as earlier – performed on the following 4 systems:

- **Ant** – a command line tool for building Java applications.[1]
- **Gremon** – a proprietary greenhouse work-flow monitoring system.[2]
- **Struts 2** – a framework for creating enterprise-ready Java web applications.[3]
- **Tomcat** – an implementation of the Java Servlet and Java Server Pages technologies.[4]

Table 3 shows some basic properties of these systems.

Table 3. Analyzed Systems

	Ant	**Gremon**	**Struts 2**	**Tomcat**
Number of developers	37	13	26	15
Maximum logical lines of code	106,413	55,282	152,081	46,606
Number of commits	6,102	1,158	1,749	1,292
Maintainability increases	1,482	456	498	269
Maintainability no change	3,051	365	710	704
Maintainability decreases	1,569	337	541	319

To provide an overview about some interesting aspects of the analyzed systems we present a couple of diagrams. First let us consider Figure 1. The small empty circles on this strip chart represent the commits in a system. On the y-coordinate the developers are listed in a decreasing order according to their number of contributions. The topmost developer is always the one with the largest contribution. On the left of the diagrams the user IDs of the developers are displayed. In case of Gremon – as it is an industrial project – the real user IDs are masked. On the right of the diagrams the portions of the total contributions are displayed. The x-coordinate represents the revisions of a system.

The black lines are actually several empty circles over one another; those are the periods when the developer in question was the most active.

Figure 2 illustrates the contributions of the authors from two aspects: the number of commits per developer, and the number of touched files per developer.

The diagrams in the first row contain the number of commits per author in a descending order. In case of Tomcat more then 80% of the commits were commited by a single developer. In projects Ant and Gremon there is again a

[1] http://ant.apache.org
[2] http://www.gremonsystems.com
[3] http://struts.apache.org/2.x
[4] http://tomcat.apache.org

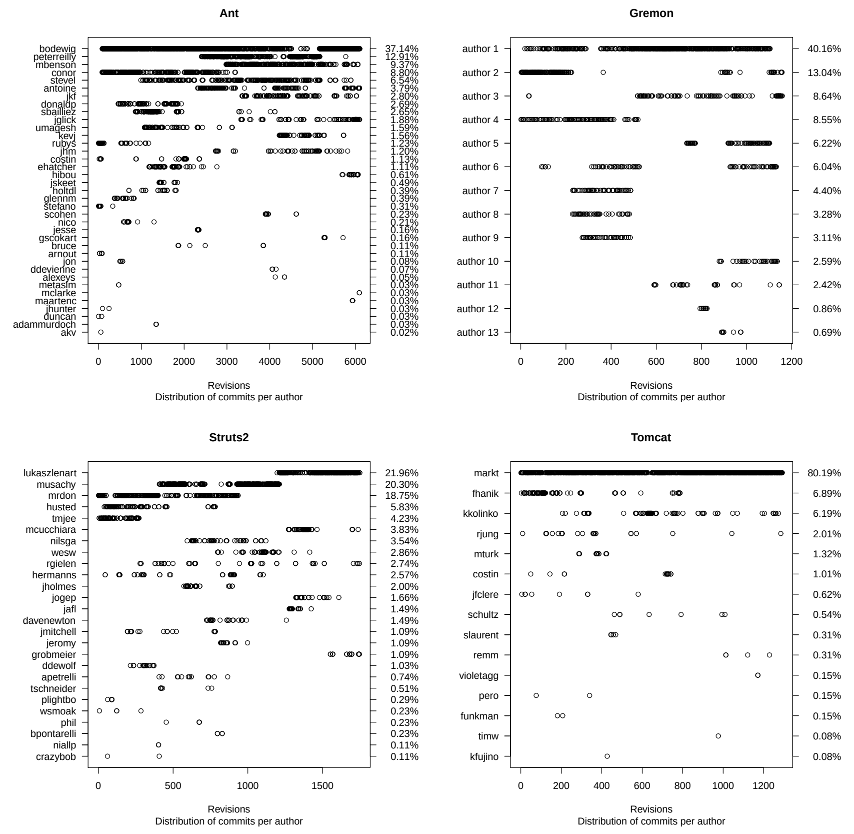
Fig. 1. Commits per authors

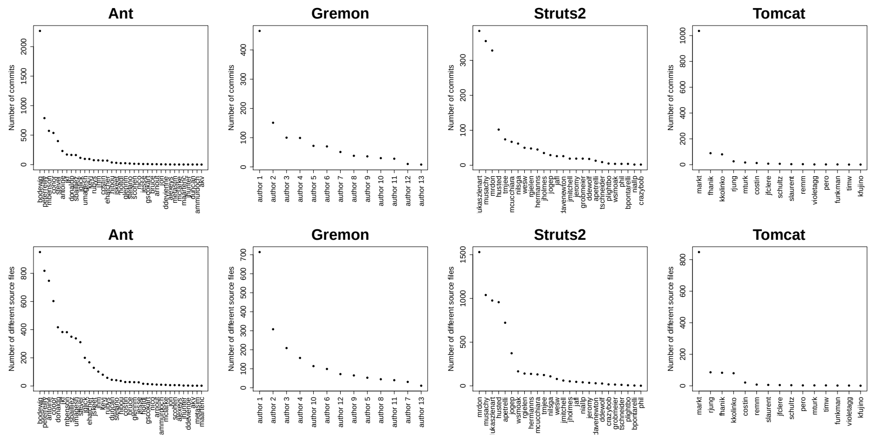
Fig. 2. Contributions per author

clear single main developer who performed about 40% of the total commits. On

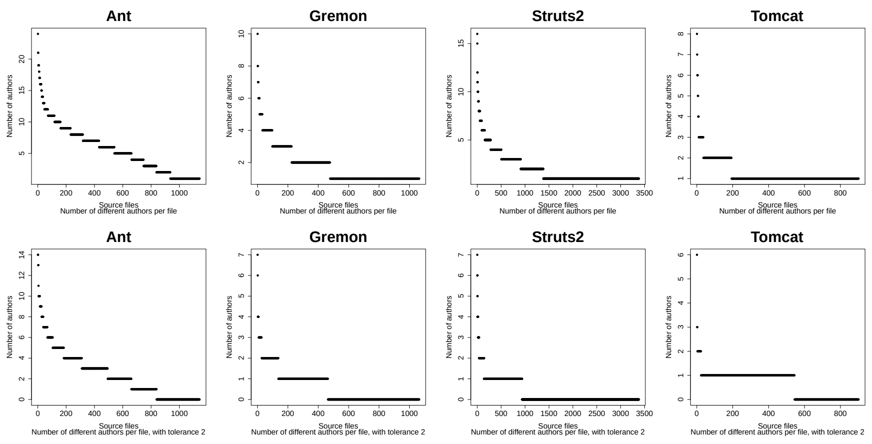

Fig. 3. Number of authors per file

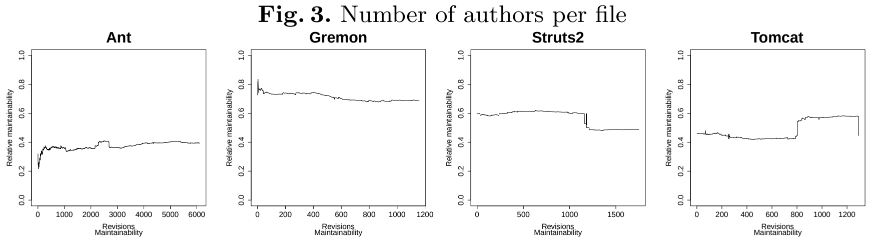

Fig. 4. Maintainability change

the other hand, in case of Struts 2, it seems that there are 3 main contributors with a more or less similar impact on the project. From the strip chart it seems that 2 of them were main developers mainly in parallel, and then the third one took over the responsibility.

In the second row we show how many files a developer commited at least once. For example, if the small black circle above a developer is at the height of 100, it means that the developer commited in 100 different files. In case of Gremon and Tomcat the domination of the main developer is obvious from this diagram as well, but in case of Ant and Struts 2 it seems that several developers have a contribution affecting a large amount of files.

Figure 3 illustrates the number of different developers per files, which can be thought as a kind of ownership. The first row of this diagram is the inverse of the second row of Figure 2, namely how many different developers commited to a single file. The black circles seem to be lines here as there are many files with the same values. If the lines at the lower values are longer, that indicates clearer separation of responsibility. Higher values indicate the hot areas: these are the files that were modified by several developers.

The second row of this diagram is similar to the first one, but it contains a relaxation: the commits of a developer is not counted here if the number of contributions of that developer on the source file in question was at most 2. We applied this rule because we wanted to eliminate the possible bias caused by a directory rename or a branch merge for example, which affects several source files by the contributor without real modifications of the source code. On these diagrams lines at 0 also appear, e.g. containing those sources which have been added once but never modified.

It is spectacular that the separation of responsibility is the best – based on the earlier statistics not surprisingly – in case of Tomcat. The separation of responsibility in case of Struts 2 and Gremon seems sufficient, but in case of Ant it is spectacularly bad. As the number of commits in this project is higher than the total number of commits in the rest of the 3 projects all together, we checked if such mess in separation of responsibility is caused just because of the long revision history or this is a true tendency. We took the first 2,000 commits and found that the same lack of responsibility separation existed even considering similar magnitude of commits as in the other 3 cases.

In Figure 4 the change of maintainability values over time is displayed. The maintainability of Ant is the lowest in overall. Not to forget that the separation of responsibility is practically missing in case of that project. The maintainability of Gremon is the highest, and in case of Struts 2 and Tomcat it is somewhere in the middle. It is worth to mention that in case of Struts 2 the maintainability decreased and in case of Tomcat it increased over time.

4.2 Results of the Statistical Tests

Table 4 shows the results of the Wilcoxon rank test described in Section 3.6. The results vary from very strong to not significant, but neither of them contradict.

Table 4. Results

System	**p-value**	**Significance**
Ant	0.033728	strong
Gremon	0.059604	significant
Struts 2	0.000014	very strong
Tomcat	0.213841	not significant

The result for project Ant is solid, above 0.01 but below 0.05. Indeed, the result for the first 2,000 commits is 0.002897, which is even below 0.01. It seems that the results weaken in the later phase of the project, when already too many sources have been contributed by too many developers.

The results for project Gremon is somewhat above 0.05, but still significant.

On the other hand, results for Struts 2 is very strong.

The tests for Tomcat show absolutely no significance; however, the results are not contradicting either. The reason for this might be the fact that more than 80% of the commits were performed by the same author, which causes a huge bias compared to the other projects. Therefore Tomcat is an atypical project from this respect.

4.3 Discussion

For proper interpretation of the above results we address some possible misinterpretations.

One could simplify the method as follows: the more people work on a system, the more complex it will be, and the more complex a systems is, the harder to maintain it. We consider this relationship as already known and did not even check it. On the other hand, we state that the effect of the *future* modifications on source files changed by more developers in the *past* is more likely to lower the maintainability compared with modifications on files that have been changed by less number of developers earlier. Note that this is not trivial: a source file with several earlier contributor is likely to be more complex than those with clear ownership, and our statement is that the already low quality source code is more likely to become even worse than that of higher quality.

For the sake of better understanding we interpret our result as follows: source files which have been modified by more developers in the past is more likely to become more complex in the future than those with less number of contributors. An even more precise statement would be the following, which is harder to conceptualize: the number of earlier contributors of modifications resulting in code quality increase is more likely to be lower than those resulting in code quality decrease.

Now we highlight the limitations of the results. It would be an inappropriate interpretation that the quality of the code with clear ownership increases, and the quality of common code decreases. There are quality decreases in the sources with clear ownership, and quality increases even in the hottest code areas. The presented results are much more modest, but significant.

Regarding the strength of the results, we stress that the 4 analyzed systems have been fixed before the case study. These are the same as we used in our earlier studies ([9], [8], [7], and others under development).

5 Threats to Validity

The factors which might threaten the validity of the results are the following.

The significance of the results are varying from very strong to not significant. We consider the system with a not significant (but not contradictory) result as an extraordinary one, as more than 80% of the commits were performed by a single developer.

We eliminated the commits related to non-traceable maintainability changes. The cardinality of these operations is between one third and one half of the total number of commits. This is a relatively large amount of data to be excluded. An enhanced model considering also the commits with no maintainability change could provide other results. However, we do not expect a different final conclusion even in the case of such a model.

There are several quality models, and there is no such unique model which is accepted by the whole industry. Using another model could provide a different result. We know that no quality model (including the one we used) is perfect. However, as most product quality models rely on a similar source of information (i.e. source code metrics) we do not expect that the results are so much dependent on the actual quality model used.

The ownership values are based on files, but the maintainability data are calculated for commits. Therefore we needed a conversion of ownership values from files to commits. We found the geometric mean to be an adequate approach; however, one could argue that we should use another aggregating method (e.g. taking the mean, the median, the maximum element, or using a more sophisticated approach). We run some tests using another aggregation method, but found no significant difference in the results of the statistical tests.

For the calculation of ownership values of a file we considered the most straightforward approach: the number of developers modifying the file so far. More sophisticated methods like introducing a tolerance (considering a contributor only above a certain threshold), or using relative basis instead of an absolute one could further enhance the methodology and the results could be more adequate.

6 Conclusions and Future Work

In this study we investigated if different number of developers of source code have an impact on the maintainability change of future commits.

We estimated the relative maintainability of every revision of four analyzed software systems, using the Columbus Quality Model, which compares the metrics of the analyzed system with the metrics of software systems found in a benchmark, and with the help of these results we identified if the net quality change caused by the actual commit was positive, neutral or negative. Additionally, we defined the source code related ownership value to be the number of different contributors so far. We aggregated the file related values to estimate a commit based ownership value by calculating the geometric mean of the file related values affected by the actual commit. Then we divided the commits into 3 sets based on the sign of maintainability change caused by the commit, therefore getting sets of commits related to positive, neutral and negative maintainability change. We omitted the commits related to neutral maintainability changes, then we took the already calculated ownership values of the two remaining subsets.

We tested the straightforward null-hypothesis regarding these ownership values that any difference in the distribution of them is casual, as the ownership value was calculated using historical data from version control system, and the maintainability change is affected solely by the actual commit. Furthermore, it considers the source code only and no other information like the developer itself. We, on the other hand, by executing Wilcoxon rank correlation test, found the following significant tendency: the ownership values related to positive maintainability changes are more likely to be lower than those related to negative maintainability changes.

We executed the test on 4 different systems selected in advance. The test for one of them resulted a very strong (p-values 0.000014), one a strong (0.034), the third one a significant (0.060) and finally the fourth one a not significant (0.21) outcome. We are convinced that the system with low significance (but not contradicting) result is an exceptional one, as more than 80% of the commits were performed by a single developer. Considering the results we can conclude that common code is more likely to erode further than code with clear ownership.

A practical use of this result could be the following. The efforts which can be spent on code quality is typically very limited. This result (and other results of similar research) can help in prevention, and in more efficient allocation of these efforts. As a prevention it is proposed that the boundaries of responsibilities within a software system should be as clear as possible. If this rule has been already somehow broken, it is recommended to pay special attention on source files of common code in case of any modifications, e.g. by mandating more strict code review rules.

As this study is part of a longer term research, we have concrete plans regarding the future steps. In short term, we plan to investigate the effect of other information found in the version control system on maintainability, like file name, date, comment or the size of the files.

As a final step, we plan to aggregate the results, and then implement a software which identifies the hot areas of the source code of the analyzed system. An IDE plug-in, which visually marks these areas could be useful for architects, and it could automatically warn the developers.

In longer term, we plan to take other information into consideration, like information found in issue tracking systems or developer environment interactions. An even more accurate result could be obtained by considering the type of the software systems as well (e.g. standalone or client-server applications).

Moving to other languages and comparing them with Java could also be an interesting future research direction. For example, the quality model used for this research was adopted to C# by Hegedűs [13].

Our long term goal is to fine-tune the formula of code erosion as much as possible in order to understand why it happens, and with this information in hand we could give hints how to avoid it with the least additional effort.

Acknowledgment

The publication is partially supported by the European Union projects titled "Telemedicine-focused research activities on the field of Mathematics, Informatics and Medical sciences", project number: TÁMOP-4.2.2.A-11/1/KONV-2012-0073 and "REPARA – Reengineering and Enabling Performance And poweR of Applications", project number: 609666.

References

1. Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R., Gyimóthy, T.: A probabilistic software quality model. In: Software Maintenance (ICSM), 2011 27th IEEE International Conference on. pp. 243–252. IEEE (2011)
2. Bakota, T., Hegedűs, P., Ladányi, G., Körtvélyesi, P., Ferenc, R., Gyimóthy, T.: A cost model based on software maintainability. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on. pp. 316–325. IEEE (2012)
3. Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B.: Does distributed development affect software quality?: an empirical case study of windows vista. Communications of the ACM 52(8), 85–93 (2009)
4. Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P.: Putting it all together: Using socio-technical networks to predict failures. In: Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on. pp. 109–119. IEEE (2009)

5. Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code!: examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. pp. 4–14. ACM (2011)
6. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. Software Engineering, IEEE Transactions on 20(6), 476–493 (1994)
7. Faragó, C.: Variance of the quality change of the source code caused by version control operations. In: The 9th Conference of PhD Students in Computer Science (CSCS) (2014)
8. Faragó, C., Hegedűs, P., Ferenc, R.: The impact of version control operations on the quality change of the source code. In: Computational Science and Its Applications–ICCSA 2014, pp. 353–369. Springer (2014)
9. Faragó, C., Hegedűs, P., Végh, Á.Z., Ferenc, R.: Connection between version control operations and quality change of the source code. Acta Cybernetica 21, 585–607 (2014)
10. Fritz, T., Murphy, G.C., Hill, E.: Does a programmer's activity indicate knowledge of code? In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 341–350. ACM (2007)
11. Gyimóthy, T., Ferenc, R., Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. Software Engineering, IEEE Transactions on 31(10), 897–910 (2005)
12. Hattori, L., Lanza, M.: Mining the history of synchronous changes to refine code ownership. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. pp. 141–150. IEEE (2009)
13. Hegedűs, P.: A probabilistic quality model for C# – an industrial case study. Acta Cybernetica 21(1), 135–147 (2013)
14. ISO/IEC: ISO/IEC 9126. Software Engineering – Product quality 6.5. ISO/IEC (2001)
15. LaToza, T.D., Venolia, G., DeLine, R.: Maintaining mental models: a study of developer work habits. In: Proceedings of the 28th international conference on Software engineering. pp. 492–501. ACM (2006)
16. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the apache server. In: Proceedings of the 22nd international conference on Software engineering. pp. 263–272. Acm (2000)
17. Nordberg III, M.E.: Managing code ownership. Software, IEEE 20(2), 26–33 (2003)
18. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2013), http://www.R-project.org/
19. Rahman, F., Devanbu, P.: Ownership, experience and defects: a fine-grained study of authorship. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 491–500. ACM (2011)
20. Weyuker, E.J., Ostrand, T.J., Bell, R.M.: 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)