# Ownership and Defects in Open-Source Software

Eric Camellini
Delft University of Technology
Delft, The Netherlands
E.Camellini@student.tudelft.nl

Kaj Dreef
Delft University of Technology
Delft, The Netherlands
K.Dreef@student.tudelft.nl

## ABSTRACT

<still missing>

## CCS Concepts

•**Software and its engineering** → **Software defect analysis;** *Open source model;* •**Information systems** → Data mining;

## Keywords

Ownership; Software quality; Process metrics

## 1. INTRODUCTION

Software defects correction has a great impact on the economy [1]. For this reason, in recent years, a wide number of studies focused on defining software metrics that can be useful to build models for *defect prediction*. A *software metric* is a measure of a property of the software that can be related to the code (*code metric*) or to its development process (*process or change metric*). Previous research also shows that process metrics are more effective when used to build a defect prediction model [7, 11]. This past results show that the developers behaviour has more impact on software quality then the characteristics of the software itself. *Code ownership* metrics are process metrics, and can in general be defined as a measurements of the contribution of a developer to a software artifact over a certain period of time.

Ownership metrics are highly dependant on the process used to develop the software and on the organizational structure of the team, because of that previous studies show contrasting results. Bird et al. [2] showed that there is a significant correlation between ownership and number of defects in the considered software artifacts in Microsoft Windows projects, while accordingly to Foucault et al. [3] the same metrics do not result in the same kind of relationship when computed on Open-Source Software artifacts.

To obtain more generalizable results ownership metrics should be computed in a way that adapts to the structure of the project. Foucault et al. [3] also introduced the idea of computing ownership metrics on artifacts of different granularities (Java files and packages). Greiler et al. [4] applied the same principle on different Microsoft Products, and their results showed that can be more significant to use ownership metrics to classify defective and non-defective artifacts rather than to try to infer the number of defects.

Unfortunately, to our knowledge, a classification approach like the one described above has not yet been applied to Open-Source Software. Furthermore, none of the previous studies tried to compute the ownership metrics on the revision of the artifact where the defective code is introduced. In addition to that, all the previously cited works computed the ownership metrics considering as variable the commit count on the software artifacts, and tried to change the granularity of the artifacts themselves, but not the granularity of the considered variable (e.g. considering the amount of lines added and deleted in every commit instead of the commit count).

Our idea is that in order to build a defect prediction model the ownership metrics should be computed on the releases of software artifacts that succeed the introduction of defective code. To determine when defective code is introduced we use the concept of *implicated code* as described by Rahman er al. [10] (also called *fix-inducing* code [12]). In this work we study the effect of code ownership on software quality by trying to classify defective files on different Open-Source Software projects. We do so by computing different ownership metrics on the files just after some implicated code is introduced, Also we experiment with the granularity of the variable used for the metrics computation so we can observe the effect of changes that it introduces on the correlation with implicated code.

<MISSING: summary of experimental results>

<MISSING: outline of the paper Sections>

## 2. THE PROBLEM

The general problem that we target in this paper is that building defect prediction models is a challenging activity, and one of its main difficulties is that it is highly project-dependant [13]. In particular our work addresses the more specific problem that currently it is difficult to generalize the effect of code ownership on software quality. Previous studies show contrasting result when trying to correlate these two aspects [2, 3, 4].

What makes the problem tricky is that code ownership highly depend on the organizational structure of the development team and on the developers behaviour. Previous



*The 13th Working Conference on Mining Software Repositories '16 Austin, Texas USA*



DOI:

studies always computed the code ownership process metrics without adapting them to the specific characteristics of the considered software project, and without taking into account that to find a relationship between these metrics and software defects their computation should be performed on the revision of the software that captures its state just after the introduction of these defects.

All the cited studies considered only the commit count to measure the proportion of ownership of a developer, but not in all the projects the commits can be considered equal, and maybe it makes sense, for example, to use a more fine-grained approach. Furthermore, think about a study that computes the ownership metrics on the artifacts of a specific software release, considering an artifact as defective because a bug affects it in that release: this bug could have been introduced way before the release, but was discovered only after it. That's why we think that it could make more sense to go back to the version of the artifact that really represent the defect introduction and then compute the ownership metrics.

# 3. BACKGROUND

## 3.1 Ownership

*Code ownership* can in general be defined as the contribution of a developer to a software artifact over a certain period of time. It describes whether the responsibility for a certain software artifacts is spread around many developers, or if there is a single person that can be considered its "owner". Ownership can be interpreted as the *responsibility*. Past studies showed that, on Microsoft products, code artifacts without a clear responsible are more defect prone [2, 4]. In these studies ownership is also interpreted as a measure of the *expertise* of a developer with respect to the code artifact.

### 3.1.1 Ownership metrics

We base the ownership metrics that we consider in this work on the a set of metrics introduced by Bird et al. [2] and then reused and revisited in further studies [3, 4]. These metrics refer to a software artifact (e.g. a file) and are:

- *Ownership*: proportion of ownership for the highest contributor;
- *Minors*: Number of contributors with a proportion of ownership that is below a certain threshold;
- *Majors*: Number of contributors with a proportion of ownership that is above the threshold for the minor;
- *Total*: The total number of contrubutors.

For these metrics the proportion of ownership can be measured using different variables: all the cited past studies used the number of commits to the artifact [2, 4, 3]. Different threshold were used to distinguish minor and major contributors: 5% [2, 3], 20% and 50% [4].

## 3.2 Implicated code

We refer to the concept of *implicated code* as described by Rahman et al. [10]: "Implicated code is code that is modified to fix a defect". Rahman et al. also describe a technique to identify the implicated code and the revision where it was introduced. The steps to do it are the following:

1. Identify a commit that fixes a bug: this commit changes one or more files from revision $n$ to revision $n + 1$;
2. Identify the lines that are deleted or changed by this commit using the *git diff* command between the revisions $n$ and $n + 1$;
3. Checkout revision $n$, since it still contains the defective code, and use the *git blame* command to identify in which revision it was introduced: the code in that revision is the implicated code;
4. If the code was introduced after the fixed bug was reported, then mark it as *innocent* (not implicated).

# 4. METHODOLOGY

Ownership of a file has been previously only been determined considering a specific release of the software project considered. This results in a situation where the ownership metrics are correlated with the defects, but are not computed when the defects are introduced. To solve this problem we want to distinguish the software revisions that introduces bugs, and to compute the metrics on the software artifacts as soon as they become defective.

In this work we will focus on computing the ownership metrics considering the files as target artifacts. We are interested in the revisions of the files that succeed the introduction of defects, so that contain implicated code: we will call these files, in the described revision, *implicated files*.

What we want to do is to identify all the implicated files in the history of a software projects, and to check if it is possible to distinguish them using the ownership information.

Given a software project, our approach works in the following steps:

1. from the project code and bug repositories we extract a data set that describes its whole history (commits, bug fixes, implicated files etc.);
2. we use this data set to compute the ownership metrics for every revision of every file;

we use this last data set to perform the defect prediction with the aim to distinguish the implicated files revisions from the non-implicated ones.

## 4.1 History data set

In order to extract the whole history of a software project we go through all the commits in its *git log*, in chronological order, and commit by commit we save information on the contribution of every author to every file. More specifically, for every commit we:

1. extract the list of the affected files;
2. for each affected file update the information about the contribution that the author of the commit performed to it in terms of lines added, lines deleted and number of commits;
3. extract information about the bugs related to the commit. In particular: if it is a bug fix (see Section 4.2);
4. add to the data set, for each one of these files, as many lines as the number of authors that contributed to it until the considered commit (included): each one of

these lines contains the information about the contribution of the corresponding author to the corresponding file, at the moment of the corresponding commit.

A complete list of the columns can be found in Table 1. Some of them contain information that are not related to the single author, but to the file or the commit: in this case their value will be the same for more than one line.

To check if a file in the revision that results from the commit of its line is implicated, we use the approach described in Section 3.2, without considering the last step: this because we assume that if the lines are added after the issue reporting date, they are still based on a defective file, and so they contribute to the defect.

Since we need to know which commit fix bugs, this information about the implicated files is added to the history data set after having build all the rest of it.

## 4.2 Bug-fixes and commit linking

<MISSING: description of the bug-code repository linking technique use to determine when a certain commit fixes a bug>

## 4.3 Ownership metrics

Based on the created history data set we compute multiple ownership metrics for every revision of every file. This will produce another data set with fewer lines (since the metrics are computed grouping by file and commit). A list of the metrics is shown in Table 2, where we refer to the terms *ownership*, *minor*, *major* and *total* as described in Section 3.1.1.

### *4.3.1 Ownership threshold*

<MISSING: description on how we select the threshold to distinguish minor and major contributors>

## 4.4 Statistical approach

The features will be used to see if there is a correlation between them and the fact that a file is implicated. For the classification we will use the *Random Forest* classification algorithm.

# 5. EVALUATION

This section will explain what are research questions that we want to answer and what methods we will use to evaluate our results, and to check if we can provide significant outcomes. We will also take a look at the projects that we considered as subjects for the study.

## 5.1 Research Questions

For this research we address 3 main research questions.

1. Are ownership metrics correlated to code quality for Open-Source Software?
2. Which level of granularity (line-based or commit-based) is better to calculate the ownership metrics on a file, in order to obtain the best correlation with the code quality (measured as the presence or absence of implicated code in the file itself)?
3. Can the threshold used to distinguish minor and major contributors be determined using project specific characteristics?

The first research question is a more general question, which is needed to figure out if in general ownership metrics can be a good indication of code quality on open-source software. The second research question is important to understand what type of granularity is better for ownership. In previous research ownership has always been related to the amount of commits done by a specific contributor, are there other ways to compute ownership that give a better result in terms of relationship with defective code? The last research question focuses on the threshold to determine major or minor contributors. In previous research arbitrary values had been chosen for the threshold, but this values does not per se work for every project, because the distinction it can depend on the size of the development team working on it, in the average size of a commit and on other factors.

## 5.2 Study Subjects

For this research we had multiple study subjects, see Table 3. All of them are open-source projects with git[1] as their version control system, and use JIRA [2] as their issue tracker. Every project contains several tens of thousands of commits. In Table 3 we show some information on every considered project.

<MISSING: Still haven't done this yet, but our goal is to do it on multiple projects>

## 5.3 Data set

<MISSING: description (e.g. in terms of size) of the resulting data sets (history and metrics)>

## 5.4 Metrics Correlation

To be able to answer the research questions we need to determine if there is a correlation between the ownership metrics and the fact that a file is implicated. This is done by finding a correlation between ownership on different granularities and implicated files. This test should give us answer on if there is a correlation between code quality and ownership metrics, and which level of granularity gives the best results.

## 5.5 Bug Classification

Classification is the last step that will be done to see how accurate the prediction can be if you classify files into defective or not based on ownership metrics. This experiment is done for two reasons. First, to get insight on how accurate a defect prediction model can be if based on ownership metrics. Second, by comparing the results obtained using different thresholds to distinguish minor and major contributors we can determine if there is a more accurate, project-dependant way to determine that threshold. Before using the the metrics for the classification we need to filter out the redundant one, meaning the ones that have high correlation with each other. The remaining metrics will then be used as features for the classification algorithm.

<MISSING: Find a good way to determine how to evaluate the chosen threshold>

<STILL NEED TO DETERMINE TYPE OF CLASSIFICATION BEING USED, NOW ASSUME RANDOM FOREST ALGORITHM>

[1] http://git-scm.com/

[2] https://www.atlassian.com/software/jira

| Column | Description |
|---|---|
| | Project name |
| | File name |
| | commit SHA |
| | Author |
| | Total lines added by the author to the file after the commit |
| | Lines added by the author to the file with the commit |
| | Total lines added to the file after the commit |
| | Total lines deleted by the author to the file after the commit |
| | Lines deleted by the author to the file with the commit |
| | Total lines deleted to the file after the commit |
| | Total commits to the file by the author after the commit |
| | Total commits to the file |
| | Commit date |
| bug_fix | Contains 1 if the commit fixes one or more bugs, zero otherwise |
| affected_versions | List of all the project releases affected by the bugs fixed by the commit |
| implicated | Contains 1 if the file is implicated after the commit |

**Table 1: Columns of the history data set**

| Feature | Description |
|---|---|
| Commit ownership | *Ownership*, computed using the commit count |
| Commit minor | *Minor*, computed using the commit count |
| Commit Major | *Major*, computed using the commit count |
| Line ownership (lines-added) | *Ownership*, computed using the amount of lines added |
| Line minor (lines-added) | *Minor*, computed using the amount of lines added |
| Line major (lines-added) | *Major*, computed using the amount of lines added |
| Line ownership (lines-deleted) | *Ownership*, computed using the amount of lines deleted |
| Line minor (lines-deleted) | *Minor*, computed using the amount of lines deleted |
| Line major (lines-deleted) | *Major*, computed using the amount of lines deleted |
| Total | *Major* number of contributors (it does not depend from the granularity) |

**Table 2: Ownership metrics computed based on the history data set**

# 6. THREATS TO VALIDITY

# 7. RESULTS

# 8. RELATED WORK

A number of prior studies focused on code ownership and its relationship with software quality. Bird et al. [2] first examined this topic defining the concepts of ownership as proportion of contribution, and of minor and major contributors. These are the concepts that we use in this work, but with a different granularity and on different type of software artifacts. Their results shows that if a Microsoft Windows code artifact does not have a well defined owner then it is more defect prone, and the same holds if a lot of minor contributors have worked on it.

Rahman et al. [10] examined the effects of ownership and authorship on software quality using a fine-grained approach and computing their metrics on chunks of implicated code. Our approach to determine software artifacts that contain implicated code is based on that work. They report findings similar to the ones reported by Bird et al. and described above. However, they consider ownership in a different way and use different metrics.

Focault et al. [3] replicated the study Bird et. al [2] on seven open-source projects, but using the same granularity for the metrics and the same threshold to distinguish minor and major contributors, and changing only the code artifacts on which the study was focused (Java files and packages). The outcome is contrasting with the previous results, it shows no strong correlation between ownership and defects, but it states that it is more significant when the metrcs are computed on more coarse-grained artifacts.

Another replication of the study from Bird et al. was recently performed by Greiler et al. [4], including also the intuition from Focault et al. of changing the granularity of the code artifacts: they used folders and files. This study was again targeted on Microsoft projects and confirms the result of [2]. They show that it is also possible to classify with a high precision defective files using the ownership metrics.

None of the studies that consider ownership as intended in this work [2, 3, 4] tried to compute it on artifacts that contain implicated code.

The concept of implicated code was used also by more previous works, for different purposes and with different names. Changes that introduce code that causes a fix are called fix-inducing by Sliwersky et al. [12] and dependencies by Purushothaman et al. [9].

A number of prior studies also tried to use a different granularity to compute the ownership metrics or to perform bug prediction using line-based approaches: Munson et al. [8] introduced the concept of code churn as a measure of code line changes, Meng et al. [6] considered fine-grained code changes over-time to measure the authorship in an accurate way and Hata et al. [5] computed ownership at a method

| Project | LOC | Implicated files | Contributors | Commits |
|---|---|---|---|---|
| Lucene | 252644 | 32250 | 59 | 30551 |

**Table 3: Characteristics of the project we considered**

level for bug prediction.

<MISSING: related work about bug-linking with commits or revisions, other related work if needed>

# 9. CONCLUSION & FUTURE WORK

# 10. ACKNOWLEDGMENTS

# 11. REFERENCES


[1] Software errors cost u.s. economy $59.5 billion annually. http://www.abeacha.com/NIST_press_release_bugs_cost.htm. Accessed: December 11, 2015.

[2] C. Bird, N. Nagappan, brendan murphy, H. Gall, and P. Devanbu. Don't touch my code! examining the effects of ownership on software quality. In *Proceedings of the the eighth joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ACM, September 2011.

[3] M. Foucault, J.-R. Falleri, and X. Blanc. Code ownership in open-source software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 39:1–39:9, New York, NY, USA, 2014. ACM.

[4] M. Greiler, K. Herzig, and J. Czerwonka. Code ownership and software quality: A replication study. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, Piscataway, NJ, USA, May 2015. IEEE.

[5] H. Hata, O. Mizuno, and T. Kikuno. Bug prediction based on fine-grained module histories. In *Proceedings of the 34th International Conference on Software Engineering*, pages 200–210. IEEE Press, 2012.

[6] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat. Mining software repositories for accurate authorship. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 250–259. IEEE, 2013.

[7] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 181–190. IEEE, 2008.

[8] J. C. Munson and S. G. Elbaum. Code churn: A measure for estimating the impact of code change. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 24–31. IEEE, 1998.

[9] R. Purushothaman and D. E. Perry. Towards understanding the rhetoric of small changes-extended abstract. In *International Workshop on Mining Software Repositories (MSR 2004), International Conference on Software Engineering*, pages 90–94. IET, 2004.

[10] F. Rahman and P. Devanbu. Ownership, experience and defects: A fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 491–500, New York, NY, USA, 2011. ACM.

[11] F. Rahman and P. Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 432–441, Piscataway, NJ, USA, 2013. IEEE Press.

[12] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.

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