Code Ownership in Open-Source Software

Matthieu Foucault
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
mfoucaul@labri.fr

Jean-Rémy Falleri
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
falleri@labri.fr

Xavier Blanc
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
xblanc@labri.fr

ABSTRACT

Context: Ownership metrics measure how the workload of software modules is shared among their developers. They have been shown to be accurate indicators of software quality. **Objective:** As they have been studied only with industrial software systems, we replicated the study but with Java free/libre and open source software (FLOSS) systems. Our goal was to generalize an "ownership law" that would state that minor developers should be avoided. **Method:** We explored the relationship between ownership metrics and fault-proneness on seven FLOSS systems, using publicly available corpora to retrieve the fault-related information. **Results:** On our corpus, the relationship between ownership metrics and module faults is weak. In the best setting, less than half of systems exhibit a significant correlation, and in the worse setting no system at all. Moreover, fault-proneness seems to be much more influenced by module sizes than ownership metrics. **Conclusion:** Ownership results cannot therefore be generalized to FLOSS systems. Further, we also perform an in-depth analysis to understand the lack of correlation between ownership metrics and module faults. We show that with several settings, the distributions of module faults and ownership metrics prevent the finding of any correlation.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.8 [**Software Engineering**]: Metrics—*process metrics*

General Terms

Managment, Measurement

Keywords

Process metrics, code ownership, reproduction study

1. INTRODUCTION

Software metrics have been defined with the main objective of expressing quantitative measures that can serve as indicators for software quality [3]. Although they were originally defined to measure software artifact characteristics (such as the number of lines of code or the number of methods, etc.), new metrics have been recently defined to measure developer's activity [17]. The main intuition of these metrics, also called *process metrics*, is that developers habits have a deeper impact on the software quality than the intrinsic characteristics of software artifacts, as suggested by previous research [14, 18].

Among process metrics, the ones measuring code ownership are of a particular interest. They measure the level to which developers own modules of a software system. More precisely, if a developer contributes to more than 5% of a module, she is considered to be a major owner of that module. In the opposite, a developer that contributes to less than 5% is considered to be a minor owner. According to Bird et al. ownership metrics are good indicators for software quality as the more minor developers contribute to a software module, the more faults it contains [2].

Such a result may therefore be used to reorganize the development teams in order to assign major owners for each modules of a software systems and to definitively avoid minor owners. However, prior to reorganizing development teams, a stronger validation of ownership metrics has to be done, which is the purpose of this paper. First, it should be noted that the empirical validation of ownership metrics performed by Bird et al. has been only done for industrial software modules (the ones of Windows Vista and Windows 7 only). Further, the metric granularity used by Bird et al., which consists in considering binaries (i.e. `.dll` files) as modules, is not applicable to every language.

This paper aims to replicate the empirical study of ownership metrics that has been originally performed by Bird et al. [2]. We performed a similar study but considered Java free/libre and open source (FLOSS) systems rather than industrial ones. Moreover we tested ownership metrics using two granularities, which are Java packages and source code files. Our objective was initially to make a step toward the generalization of the "ownership law" stating that developers should be major owners of several modules and that minor owners should be avoided. The results we obtain unfortunately do not exhibit the existence of such an ownership law. First, we do find a correlation between ownership metrics and the module faults, but only on half of the systems, at best. Moreover, when blocking for the module sizes (i.e.

eliminating the impact of size) we did not find relationships between ownership metrics and module faults. These findings seems to indicate that code metrics are better indicator of software quality than ownership metrics on Java FLOSS systems. Finally, all these observations show that the "ownership law" cannot be generalized. Further we performed a deeper analysis of the different software systems we studied to understand why relationships between ownership metrics and module faults are not found everywhere. We notice that the distribution of ownership metrics may have an impact on these relationships.

This paper starts in section 2 with an overview of the related works. Section 3 first presents the original ownership study performed by Bird et al. and then explain the methodology of our replication study. Section 4 presents the main result of our study that is that the "ownership law" cannot be generalized to Java FLOSS systems. These results are discussed in section 5, Finally, section 6 concludes this paper.

2. RELATED WORK

In the late 2000s, several studies have shown evidence of a relationship between the number of developers of a software artifact and its fault-proneness. Among them, Weyuker et al. found that adding the number of developers who edited a file to their prediction model provides a slight improvement to the model's precision [20]. Illes-Seifert and Paech found a correlation between the number of faults identified on a file and its number of authors [10]. Furthermore, they explored the relationship between several process metrics and fault-proneness, and did not found a metric were the relationship with fault-proneness existed in all projects. However, they found that the number of distinct authors of a file was correlated to the number of faults in *almost* every project [11].

Many studies used fault prediction models to validate the relevance of process metrics for measuring software quality. Moser et al. compared the predictive power of two sets of software metrics, respectively code and process metrics, on several Eclipse projects [16]. They found that process metrics are better indicators of software quality than code metrics. Similar results have been found by other researcher, also using fault prediction as a quality indicators for their metrics [14, 18].

Finally, D'Ambros et al. evaluated different sets of metrics in a thorough study on fault prediction [5]. They compared the process metrics introduced by Moser et al. [16] to other metrics such as the classical source code metrics from Chidamber and Kemerer [3], the measure of entropy of changes introduced by Hassan [8], the churn of source code metrics or the entropy of source code metrics. Among all these measures, they found that process metrics and the churn and entropy of source code metrics are the best performers for fault prediction. However the authors expressed serious threats to the external validity of their study (i.e. whether the results are generalizable), which calls for more empirical studies on that matter.

Although the number of developers is not always the process metric showing the higher correlation, ownership metrics rely on another information which is the proportion of contributions made by the developers. Using this information it is possible to split developers in major and minor contributors. The relationship between measures of code ownership and faults have been studied by Bird et al. on Windows Vista and Windows 7 binaries [2]. The results of this study are that the number of minor contributors of a binary is strongly correlated to the number of pre- and post-release faults of Windows binaries.

3. METHODOLOGY

This section starts by presenting the original study of ownership metrics performed by Bird et al. [2]. Then it presents our replication study.

3.1 Original Ownership Study

The goal of the original study of Bird et al. was to evaluate if analyzing how many developers coded a system, and in which proportions, could influence the fault-proneness of software modules. To that extent, they performed a study were they analyzed two commercial Microsoft systems (Windows Vista and Windows 7). Their data corpus includes pre and post-release faults, precisely linked to the faulty software modules. They introduce several *ownership metrics* that characterize the way in which a software module has been built by its developers. These metrics are further described in the next section.

3.1.1 Ownership Metrics

To present how the ownership metrics are measured, we need to briefly introduce the concepts of *software module* and *developer contribution*. First of all, we consider that any software system is composed of a finite set of software modules (i.e. components, classes or functions) that are developed by a finite set of developers which submit their modifications by sending commits.

More formally, let M be the set of software modules of a given software system, D the set of developers and C the set of commits they send.

Each module is defined by a finite set of source code files. When a developer modifies one of the files of a software module by committing her work, she is contributing to that module. The contributions made by a developer to a given module can be measured with different metrics (e.g. by counting the number of modified lines of codes). In their study, Bird et al. simply measured the weight of a developer contribution by counting the number of touched files. For instance, if Alice contributes to a module by modifying three files first and five files later, she is contributing with a score of 8.

More formally, let $w(m_i, d_j, c_k) \in \mathbb{N}$ be the number of files that belong to the software module m_i and that have been modified by the developer d_j during the commit c_k. For the sake of simplicity we define w_m the sum of all developer contributions performed on a software module m, w_d the sum of all contributions performed by a developer d and $w_{m,d}$ the sum of all contributions performed by a developer d on a module m .

Based on this concept of developer contribution, the ownership metrics mainly measure the ratio of contributions made

by one developer compared to the rest of the developers. More formally, for each module m and for each developer d such a ratio is $own_{m,d} = \frac{w(m,d)}{w(m)}$.

Further, Bird et al. developed the ownership metrics by defining four scores that are computed for each software module:

Ownership This score is the highest value of the ratio of contributions performed by all developers. More formally, for a given software module m, its *Ownership* value is $\max(\{own_{m,d} | d \in D\})$.

Minor This score counts how many developers have a ratio that is lower than 5%. More formally, for a given software module m, its *Minor* value is $|\{0 < own_{m,d} \leq 5\% | d \in D\}|$. Such developers are considered to be minor contributors of the software module

Major This score counts how many developers have a ratio that is bigger than 5%. More formally, for a given software module m, its *Major* value is $|\{own_{m,d} > 5\% | d \in D\}|$. Such developers are considered to be major contributors of the software module.

Total This score is simply the total number of developers of a module m: Total = Minor + Major.

The 5% threshold used by the metrics *Minor* and *Major* has been evaluated by Bird et al. [2] with thresholds from 2% to 10%, which produced similar results.

In addition to the Ownership metrics, Bird et al. also compute several classical code metrics for each software module:

Size The number of lines of code of a module m.

Complexity Although the exact definition of the complexity metric used in Bird et al. study is not given, we choose to use the weighted method count (WMC) of a module m for this purpose.

3.1.2 Methodology and Results

Bird et al. computed the described metrics on every binary source code files of Windows Vista and 7. Since they also have the number of pre and post release faults for each binary, they compute the correlation coefficients between the metrics and the number of faults by using the non-parametric Spearman method.

To ensure that ownership metrics have a real added-value compared to classical code metrics such as *Size*, as advocated in [6], they also performed multiple linear regression and compare the results of a model using the classical metrics and a model using both classical and ownership metrics to explain the fault numbers.

Their results indicate that ownership metrics were strongly correlated with the pre-release faults, even better than the classical metrics. They also found that there is a real added-value to consider the ownership metrics in addition to the classical metrics.

3.2 Our Replication

3.2.1 Corpus

In our replication of the code ownership study, we choose seven Java FLOSS systems, shown in Table 1. The original systems chosen by Bird et al. were Windows Vista and 7 that are commercials and closed-source systems. We chose Java FLOSS systems because we believe that the development process is very different between in this kind of systems. Therefore, generalizing the results of Bird et al. on such a corpus would be a great step toward an ownership law. In the original study, Bird et al. used tools and information specific to Microsoft, which provided the links between faults and software modules. Obtaining this information is harder on Java FLOSS systems as they do not use the same conventions and fault tracking tools. To overcome this issue, we leverage on previous empirical data that provided publicly available corpora: the PROMISE corpus [15], and the corpus used in the study of D'Ambros et al. [5], available online[1]. Both corpora associate Java classes to their number of faults for specific releases of the systems. Moreover, these corpora provide code metrics including the number of lines of code of a class and its complexity, that we need for our study.

A noteworthy difference between our corpus and the one of Bird et al. is that our corpus contains post-release faults exclusively whereas the original study targeted both pre and post-release faults [2].

3.2.2 Module Granularity

The study performed by Bird et al. considers the compiled Window binaries as a software modules. In the Java world, it would correspond to consider Java classes as modules, since the Java compiler produces binaries for each Java class. A suitable approximation to Java classes are the files in which they are coded. However we think that it may be too fine-grained, because typical Java classes are much smaller than Windows binary source files. Therefore we decided to use two definitions of module in our corpus, using two granularities. Modules can either be the Java source files (called *file* granularity) or the Java packages (called *package* granularity).

3.2.3 Analyzed Time Period

Ownership metrics are computed from the modifications performed by the developers of a module. Therefore, the amount of modifications taken into account has an impact on the metrics values. In the original study, Bird et al. considered all the modifications performed since the last release of the software, which in that case was the previous version of the Windows operating system. Regarding this point, we wanted to explore what happens when we consider only the modifications performed since the previous release, and and when considering a wider period such as the whole history available in the software repository. Therefore we use two different time periods to compute ownership metrics: from the beginning (called the *whole* period), and from the last release (called the *release* period).

3.2.4 Correlation and Module Size

[1] `http://bug.inf.usi.ch/`

Table 1: The Java FLOSS systems included in our corpus

Project	Version	Date	Previous Version	Date
Apache Ant	1.7.0	2006-12-12	1.6.5	2005-06-02
Apache Camel	1.6.0	2009-02-17	1.5.0	2008-10-31
Apache Log4J	1.2.0	2002-05-10	1.1.3	2001-06-19
Apache Lucene	2.4.0	2008-10-03	2.3.2	2008-05-06
Eclipse JDT Core	3.4	2008-06-13	3.3.2	2008-01-31
Eclipse PDE UI	3.4.1	2008-09-03	3.4	2008-06-03
Eclipse Equinox Framework	3.4.0	2008-06-06	3.3.2	2008-01-18

In the original study, Bird et al. used multiple linear regression to take into account the effect of using ownership metrics in addition to classical code metrics, following the advice given in [6]. To that extent, they compare the amount of variance in failures explained by a model that includes the ownership metrics to a model that only includes a classical code metric, such as *Size*. In our study, we also measures this effect, but we use a slightly different statistical framework. Indeed, using multiple linear regression assumes normally distributed residuals. To avoid this difficulty, we use partial correlation [13]. Partial correlation aims to compute a correlation coefficient between two variables by taking into account the effect of a set of controlling variables. We compute it using the Spearman method, because it is a non-parametric test, and by taking into account the effect of the *Size* metric, which have been shown to have a strong effect on the fault-proneness [6]. Using partial correlation, we are therefore able to analyze the added value of ownership metrics compared to using only the *Size* metric.

3.2.5 Toolset

As the informations about bugs are available in the corpora presented above, all that is left to extract are the contributions of the developers. This information is available in the version control system (VCS) of each system. In order to ease the extraction of information from the VCSs, we use an open source framework dedicated to mining software repositories called *Harmony*[2][7]. This framework provides an homogeneous model for several VCSs, including Git[3] and Subversion[4], which are the ones used by the systems of our corpus. The set of Harmony analyzes performed in our study is available online[5].

4. RESULTS

As we have two factors (granularity and period) with two possibilities for each (file and package for the first one, and whole and last release for the second one), we launched our experiment four times. The results are shown in four tables: Table 2, Table 3, Table 4 and Table 5. In these tables, the *Correlation* part shows the Spearman correlation coefficients with our metrics and the module faults. This part is divided in two parts to show separately the coefficients for ownership metrics and code metrics. These tables also contain a *Partial Correlation* part where we show the coefficients of the partial correlation between ownership metrics and module faults, taking into account the effect of the *Size* metric. They are computed using partial correlation in combination with the Spearman method. For the sake of clarity, correlation coefficients are displayed in bold if their absolute values are above the 0.50 value, meaning that they show a significant correlation. Also, we show in italic with a * symbol, the correlation coefficients that were not statistically significant under the 0.05 p-value.

4.1 Ownership vs Code Metrics

The results shown in the four tables indicate that the code metrics are better correlated with the number of faults rather than with the ownership metrics. There are only very few cases where an ownership metric has a better correlation coefficient than a code metric. This is a completely different result from the Bird et al. study where they found a better correlation of ownership metrics. It therefore shows that code metrics are better indicators of fault proneness than ownership ones for Java FLOSS systems.

Among ownership metrics, the best ones are total and minor. This is a small difference from the results of Bird et al. where minor were performing the best. Therefore, on Java FLOSS systems, the number of developers seems to be an equally good indicator of fault-proneness than the number of minor developers. Regarding the code metrics, there is almost no difference between the size and complexity metrics.

4.2 File vs Package

The results for the package granularity (Table 2 and 4) are far better than the ones for the file granularity (Table 3 and 5). Using the file granularity, the results contain only one case of correlation between a metric and the number of bugs, whereas using the package granularity there are twenty of such cases. Therefore ownership and code metrics should be computed at the package granularity and not on the file granularity for Java FLOSS systems. This confirms our initial intuition that Java files are too fine-grained entities compared to Windows binaries.

4.3 Whole vs Last Release

The impact of computing ownership metrics over the last release of a project (Table 2 and 3) or over its whole history (Table 4 and 5) is not consistent across projects. With a package granularity, the results of four projects (JDT, Log4J, Camel and Lucene) are only slightly different between both periods, for all the ownership metrics. With other projects, results are contradictory. With Ant, the correlation between each ownership metric and the number of faults is stronger when considering only the last release of the

[2] `http://code.google.com/p/harmony`
[3] `http://git-scm.com`
[4] `http://subversion.apache.org/`
[5] `http://se.labri.fr/articles/ownership`

Table 2: Regular and partial correlation for the *package* granularity and *last release* period.

	Correlation						**Partial Correlation (controlled with Size)**			
	Ownership metrics				**Code metrics**		**Ownership metrics**			
Project	Ownership	Major	Minor	Total	Complexity	Size	Ownership	Major	Minor	Total
Equinox	**0.6**	**0.54**	0.32	**0.64**	**0.72**	**0.73**	0.38	0.32	*0.2**	0.41
JDT	*0.12**	-0.45	**0.77**	**0.74**	**0.85**	**0.84**	*-0.02**	-0.48	0.47	*0.28**
PDE	0.42	0.44	0.27	0.49	**0.58**	**0.61**	0.33	0.35	*0.14**	0.38
Ant	**-0.52**	-0.5	**0.67**	**0.67**	**0.63**	**0.56**	-0.34	*-0.2**	0.39	0.4
Camel	*0.16**	*0.1**	0.3	0.33	0.36	0.4	*0.08**	*0.03**	*0.13**	0.19
Log4J	*0.22**	**-0.57**	**0.59**	0.42	**0.85**	**0.9**	*0.15**	*-0.2**	*0.31**	*0.3**
Lucene	*0**	-0.24	0.49	0.41	0.47	0.49	*-0.01**	*-0.17**	0.33	0.27

Table 3: Regular and partial correlation for the *file* granularity and *last release* period.

	Correlation						**Partial Correlation (controlled with Size)**			
	Ownership metrics				**Code metrics**		**Ownership metrics**			
Project	Ownership	Major	Minor	Total	Complexity	Size	Ownership	Major	Minor	Total
Equinox	0.27	0.3	0.00	0.3	**0.52**	**0.54**	0.15	0.17	0.00	0.17
JDT	0.22	0.27	0.32	0.35	0.41	0.42	0.11	0.14	0.24	0.21
PDE	0.22	0.23	0.00	0.23	0.27	0.24	0.19	0.2	0.00	0.2
Ant	-0.32	-0.11	0.4	0.39	0.49	0.43	-0.15	-0.12	0.25	0.2
Camel	0.11	0.12	0.08	0.13	0.18	0.19	*0.05**	0.07	*0.06**	0.07
Log4J	*0.02**	0.16	0.32	0.39	0.21	0.25	*0.04**	0.15	0.27	0.35
Lucene	0.13	0.14	0.16	0.16	0.15	0.17	0.1	0.11	0.14	0.13

Table 4: Regular and partial correlation for the *package* granularity and *whole* period.

	Correlation						**Partial Correlation (controlled with Size)**			
	Ownership metrics				**Code metrics**		**Ownership metrics**			
Project	Ownership	Major	Minor	Total	Complexity	Size	Ownership	Major	Minor	Total
Equinox	-0.49	**-0.72**	**0.68**	**0.56**	**0.72**	**0.73**	*-0.17**	-0.4	0.33	*0.21**
JDT	*-0.16**	-0.33	**0.73**	**0.73**	**0.85**	**0.84**	*-0.04**	*0.07**	0.41	0.41
PDE	-0.24	-0.28	**0.54**	**0.54**	**0.58**	**0.61**	*-0.12**	*-0.1**	0.33	0.33
Ant	-0.38	*-0.19**	0.44	0.44	**0.63**	**0.56**	*-0.24**	*0.04**	*-0.05**	*-0.04**
Camel	-0.25	-0.25	0.49	0.49	0.36	0.4	-0.22	*-0.06**	0.35	0.35
Log4J	*0.23**	**-0.55**	0.49	*0.34**	**0.85**	**0.9**	*0.02**	*-0.3**	*0.38**	*0.33**
Lucene	-0.25	-0.34	0.45	0.44	0.47	0.49	*-0.17**	*-0.11**	0.24	0.26

Table 5: Regular and partial correlation for the *file* granularity and *whole* period.

	Correlation						**Partial Correlation (controlled with Size)**			
	Ownership metrics				**Code metrics**		**Ownership metrics**			
Project	Ownership	Major	Minor	Total	Complexity	Size	Ownership	Major	Minor	Total
Equinox	-0.36	-0.41	0.45	0.45	**0.52**	**0.54**	-0.25	-0.27	0.3	0.3
JDT	-0.07	-0.16	0.35	0.35	0.41	0.42	*-0.01**	*-0.05**	0.16	0.16
PDE	*0.02**	-0.18	0.17	0.16	0.27	0.24	*0.04**	-0.1	0.09	0.09
Ant	-0.13	-0.17	0.32	0.32	0.49	0.43	-0.1	*-0.03**	0.11	0.11
Camel	-0.1	-0.18	0.23	0.22	0.18	0.19	-0.1	-0.1	0.17	0.17
Log4J	-0.24	-0.31	0.38	0.34	0.21	0.25	-0.2	-0.28	0.33	0.3
Lucene	-0.12	-0.18	0.21	0.19	0.15	0.17	-0.1	-0.15	0.18	0.16

project (absolute values are at least 0.50, but at most 0.44 with the whole history). With PDE the effect is different between metrics for *Ownership* and *Major*, the correlations are stronger with only the last release, but with *Minor* and *Total* it is the reverse. Finally a major effect is found with the Equinox project and the metrics *Ownership* and *Major*, which are both positively correlated when considering the last release and both negatively correlated (i.e. the number of faults diminishes when the metric values increase) when considering the whole history of the project. Such observations are similar with the file granularity, although the correlations are a lot weaker as discussed in the previous paragraph.

Regarding these results, looking at the last release or at the whole project return either the same or contradictory results. A possible lead to solve this issue, may be to measure code ownership on both periods, last release and whole history, and to aggregate the results.

4.4 Controlling with Size

The partial correlation controlled with the module size looks for a relationship between ownership metrics and bugs while keeping a constant module size. When considering package as the module granularity (Table 2 and 4), many tests provide statistically insignificant results, due to sample sizes that were not large enough. The statistically significant results show only a low, at most moderate, relationship between ownership metrics and module faults. With the file granularity (Table 3 and 5), as there are more data points, the number of statistically significant results is higher. However the correlations are weaker than with the package granularity. As a consequence, although we used a distinct statistical test, we were not able to confirm the results of Bird et al. on Java FLOSS regarding the relationship between ownership metrics and post-release bugs when controlling the module size.

4.5 Threats to validity

Construct validity refers to whether or not our actual measures correspond to the conceptual ones. In our survey, two main concepts are measured : post-release bugs and ownership metrics. The count of number of bugs we refer to have been extracted in previous research and made public, either by Jureczko et al. [12] or D'Ambros et al. [5]. Both studies acknowledge threats to validity related to the construction of these corpora, particularly on the algorithms linking fault to their fixing commits. In both cases the algorithm consists in analyzing commit messages to find a fault identifier. When it is the case the commit is tagged as a fixing commit, and all the classes modified in this commit are linked to the fault. Although this technique represents the state of the art in the literature of linking faults to commits [21], it suffers from a poor recall as all the fixing commits which do not contain a reference to the fault are ignored. This fact threatens the validity of the study as it introduces a bias in the bugs included in the corpus [1].

Another threat related to the faults and identified by previous research is the quality of the data stored in the issue tracker itself. According to Herzig et al. [9], many issues classified as faults in open source projects are in fact evolution or optimization requests. To ensure this D'Ambros et al. have performed a manual check on one of the projects of their corpus, and found that most of the bugs were correctly classified. However we do not know if such a verification has been done on the corpus of Jureczko et al.

External validity refers to whether the results of a study are can be generalized or not. A parameter which may threaten external validity is how the subject of the study (the systems in our case) have been selected. Our corpus is composed of FLOSS systems only and the results may be different with industrial systems, who may enforce strong ownership, such as the Windows operating systems studied by Bird et al. [2]. However, as noted by D'Ambros et al, their corpus is composed of Eclipse systems, that have a strong industrial background.

The second threat to external validity is linked to the programming language and the module granularities we used. All the systems in our corpus are programmed using Java, and the package-level granularity is heavily linked to this language, as it relies on one of its feature. To resolve this issue we could include systems developed in different languages and use a language-agnostic approach to split the project into modules (e.g. [4]). Ultimately, the only way to test external validity is by replicating the study with different systems and/or different settings, to determine where an "ownership law" could apply.

Statistical conclusion validity relates to the statistical significance (or statistical power) of the results. In the case of correlation tests, the statistical power of the results is expressed with a *p-value*, representing the probability that we find a type I error (i.e. finding a correlation when none exists). When we tested for partial correlation with a package granularity, many of our tests did not provide statistically significant results due to the lack of data points.

5. DISCUSSION

Bird et al. found very high correlation coefficients between ownership metrics and module faults in industrial systems whereas we found completely different results for Java FLOSS systems. This is maybe due to the inherent differences between industrial and FLOSS systems.

One main difference between industrial and FLOSS systems is regarding workload distributions. In industrial systems, almost all developers spend 100% of their time on the system and contribute to it for several months. In FLOSS systems there are two kinds of developers. Few heroes contribute to all the modules of the system for a long time whereas there are several minor contributors who develop a single feature or fix a bug and then stop the system development [19].

We therefore investigated on the distributions of ownership metrics and module faults to check if they have an impact on the correlation coefficients. Not surprisingly, we observed that the distribution of ownership and metrics and bugs are inequitable for FLOSS systems. Rather than investigating the distributions for all granularities and for all periods, we choose to focus on the *Minor* metric, which has the best correlation among the ownership metrics. We analyse the minor developers and module faults distributions using two different settings. First we use the package modularity and

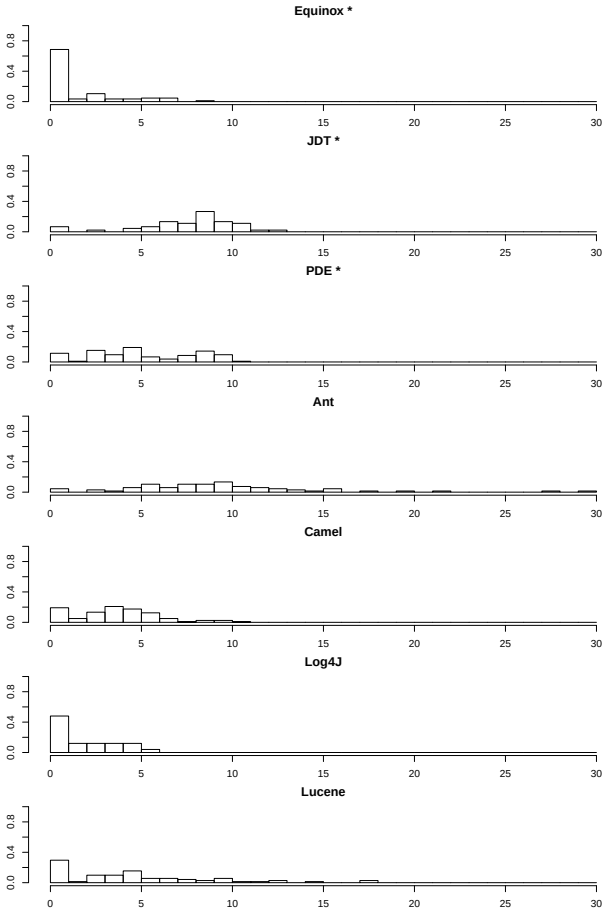

Figure 1: Distribution of the number of minor developers in the best case

whole time period since this setting produces the best correlation for the minor metric. This setting is refered to as the *best* setting. Second, we use the file modularity and the last release time period, because it produces the worse correlation for the minor metric. This setting is refered to as the *worse* setting.

Figures 1 and 2 show these distributions of the minor developers and the module in the best setting. Regarding the minor develeopers metric, its distribution is not so inequitable. Unless for the Equinox project where many packages have 0 minor contributors, the number of minor contributors per package is equitable for the other projects. In contradiction, the distribution of the number bugs is clearly inequitable as almost all of the projects contain many package without any bug.

In contradiction, the Figures 3 and 4 show the distributions of the minor developers metric and the module faults in the worse setting. These two figures clearly show that the distribution is inequitable. For all projects most of their files have no minor developers and no bugs. This is clearly a bias for measuring a correlation.

As a consequence, it appears to us that the intrinsic nature

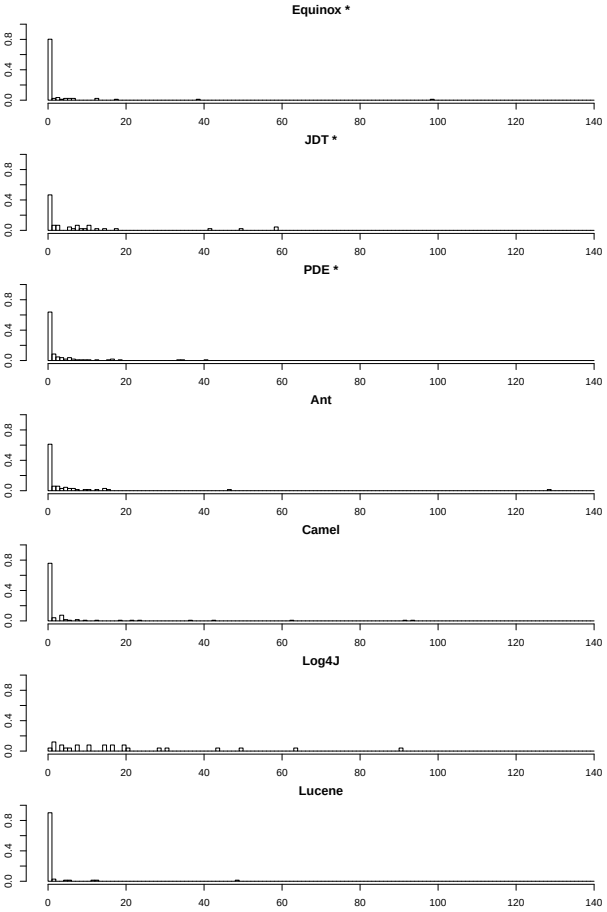

Figure 2: Distribution of the number of bugs in the best case

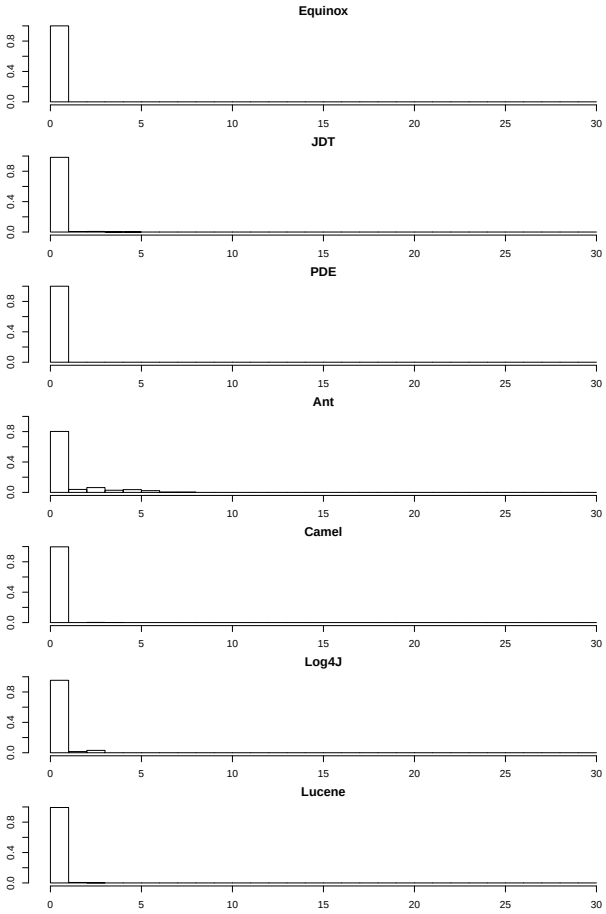

Figure 3: Distribution of the number of minor developers in the worst case

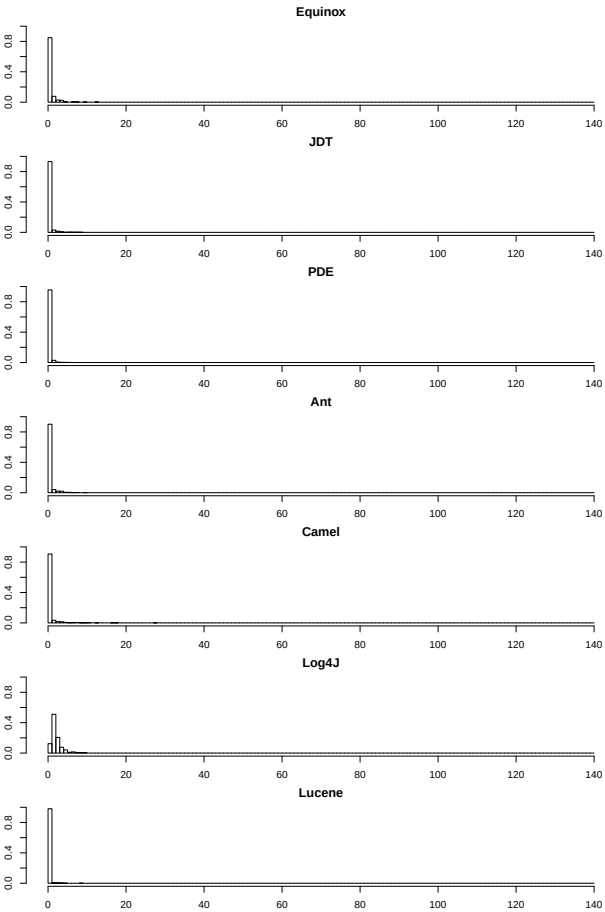

Figure 4: Distribution of the number of bugs in the worst case

of FLOSS systems is a major bias for measuring a correlation between ownership metrics and module faults. In other words, if a FLOSS systems is developed by few heroes and if lots of other developers provide few modifications then the ownership metrics are not good indicator for quality. Although this has to be checked by a complete study, we suspect that the quality of such systems is driven by the quality of the heroes.

6. CONCLUSION

The objective of our study was to replicate the study of ownership metrics of Bird et al. on FLOSS systems. Such metrics are well known to be good quality indicators and may be used to organize software development teams. Ownership metrics were validated, but only on Microsoft industrial software systems. We then choose to validate them on FLOSS systems to check if they can be applied in such a context.

Surprisingly, the results we obtain are completely different than the ones obtained in the original study. In particular, we did not observe a strong correlation between ownership metrics and module faults in the seven well known FLOSS systems of our corpus. Further, we observed that classical code metrics are better quality indicators. Moreover, no relationship between ownership metrics and module faults is found when comparing modules of similar size.

Thanks to our study, we also observed that the granularity of the software module is an important factor. The file granularity is definitively too thin. The package granularity is better but still is not good enough to observe a large correlation. We finally observed that the period of observation has a limited bug possibly strange impact on the ownership metrics.

Based on our observations we investigated the distributions of the minor developer and module faults. Our observations show that developer's workload is inequitable in FLOSS projects, which confirm well know results. When the workload is drastically inequitable, when a hero is doing almost all the job for instance, then the ownership metrics are clearly not suitable quality indicators.

7. REFERENCES

[1] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced?: bias in bug-fix datasets. 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*, pages 121–130, 2009.

[2] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. 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*, ESEC/FSE '11, pages 4–14. ACM, 2011.

[3] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transaction on Software Engineering*, 20(6):476–493, June 1994.

[4] J. F. Cui and H. S. Chae. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Information and Software Technology*, 53(6):601–614, June 2011.

[5] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.

[6] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Trans. Software Eng.*, 27(7):630–650, 2001.

[7] J.-R. Falleri, C. Teyton, M. Foucault, M. Palyart, F. Morandat, and X. Blanc. The harmony platform. Technical report, Univ. Bordeaux, LaBRI, UMR 5800, Sept. 2013.

[8] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88, 2009.

[9] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401, 2013.

[10] T. Illes-Seifert and B. Paech. Exploring the relationship of history characteristics and defect count: an empirical study. In *Proceedings of the 2008 workshop on Defects in large software systems*, page 11âĂŞ15, 2008.

[11] T. Illes-Seifert and B. Paech. Exploring the relationship of a fileâĂŹs history and its fault-proneness: An empirical method and its application to open source programs. *Information and Software Technology*, 52(5):539–558, May 2010.

[12] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, page 9:1âĂŞ9:10. ACM, 2010.

[13] M. G. Kendall and S. Alan. The advanced theory of statistics. vols. ii and iii. 1961.

[14] S. Matsumoto, Y. Kamei, A. Monden, K.-i. Matsumoto, and M. Nakamura. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 18, 2010.

[15] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

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

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

[18] F. Rahman and P. Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, page 432âĂŞ441, 2013.

[19] F. Ricca and A. Marchetto. Are heroes common in FLOSS projects? In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 55, 2010.

[20] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Using developer information as a factor for fault prediction. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, page 8, 2007.

[21] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007*, page 9, May 2007.