# Ownership and Defects in Open-Source Software

Eric Camellini and Kaj Dreef

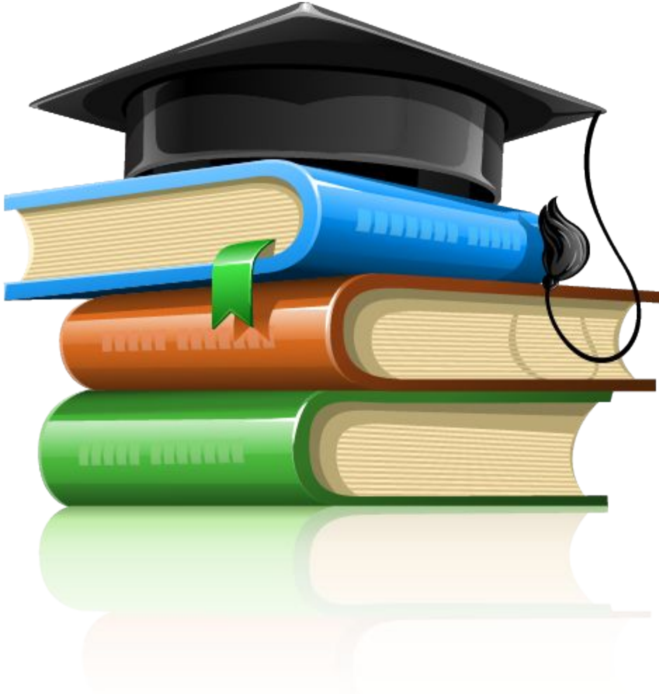
# Outline

1. Related work → Motivation
2. Proposed solution
3. Methodology
4. Evaluation & results
5. Conclusion

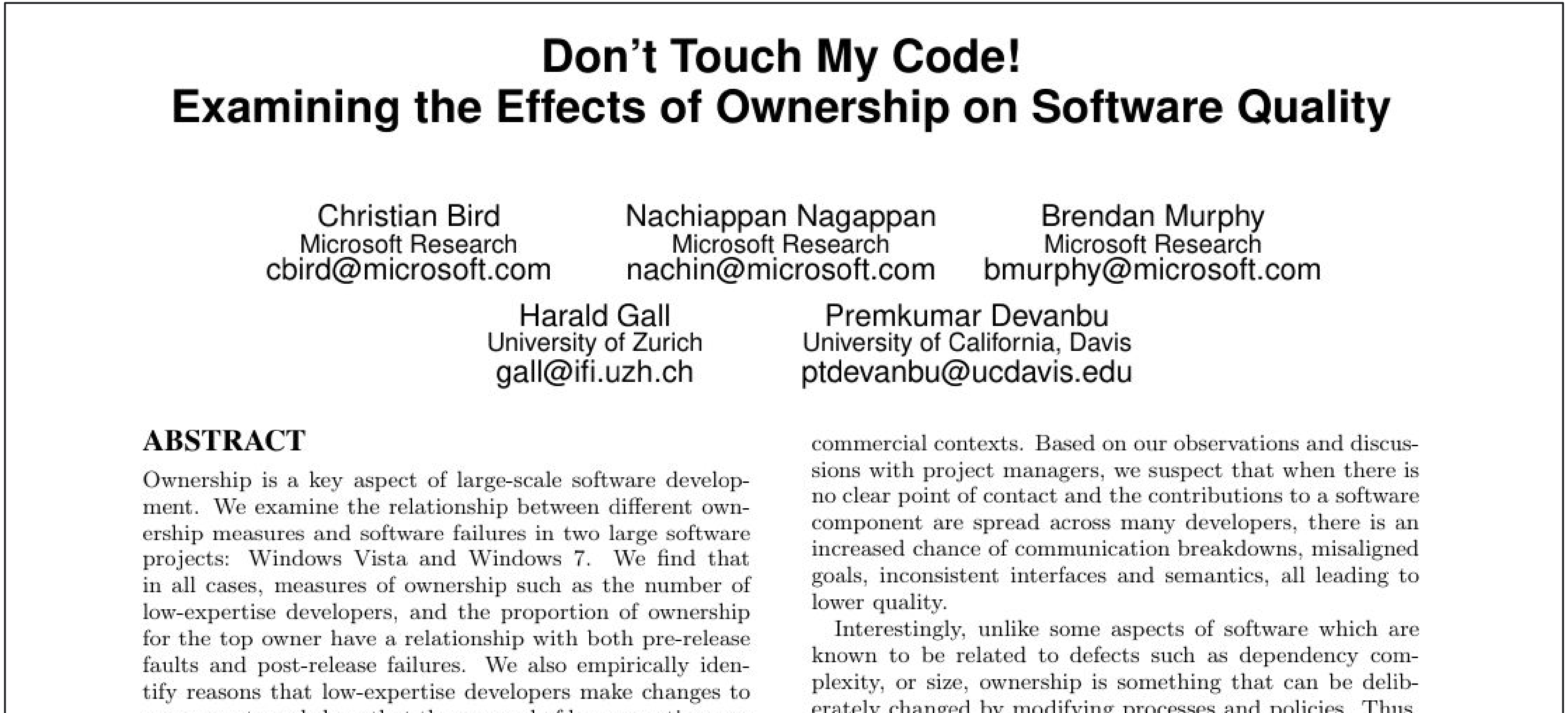
# Ownership

*A measure of the contribution of the developers to a software artifact over a certain period of time, in terms of code changes (e.g. number of commits, amount of lines added, etc.)*

# Related work
# &
# Motivation



# [1] - Ownership and n° of defects



## Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird
Microsoft Research
cbird@microsoft.com

Nachiappan Nagappan
Microsoft Research
nachin@microsoft.com

Brendan Murphy
Microsoft Research
bmurphy@microsoft.com

Harald Gall
University of Zurich
gall@ifi.uzh.ch

Premkumar Devanbu
University of California, Davis
ptdevanbu@ucdavis.edu

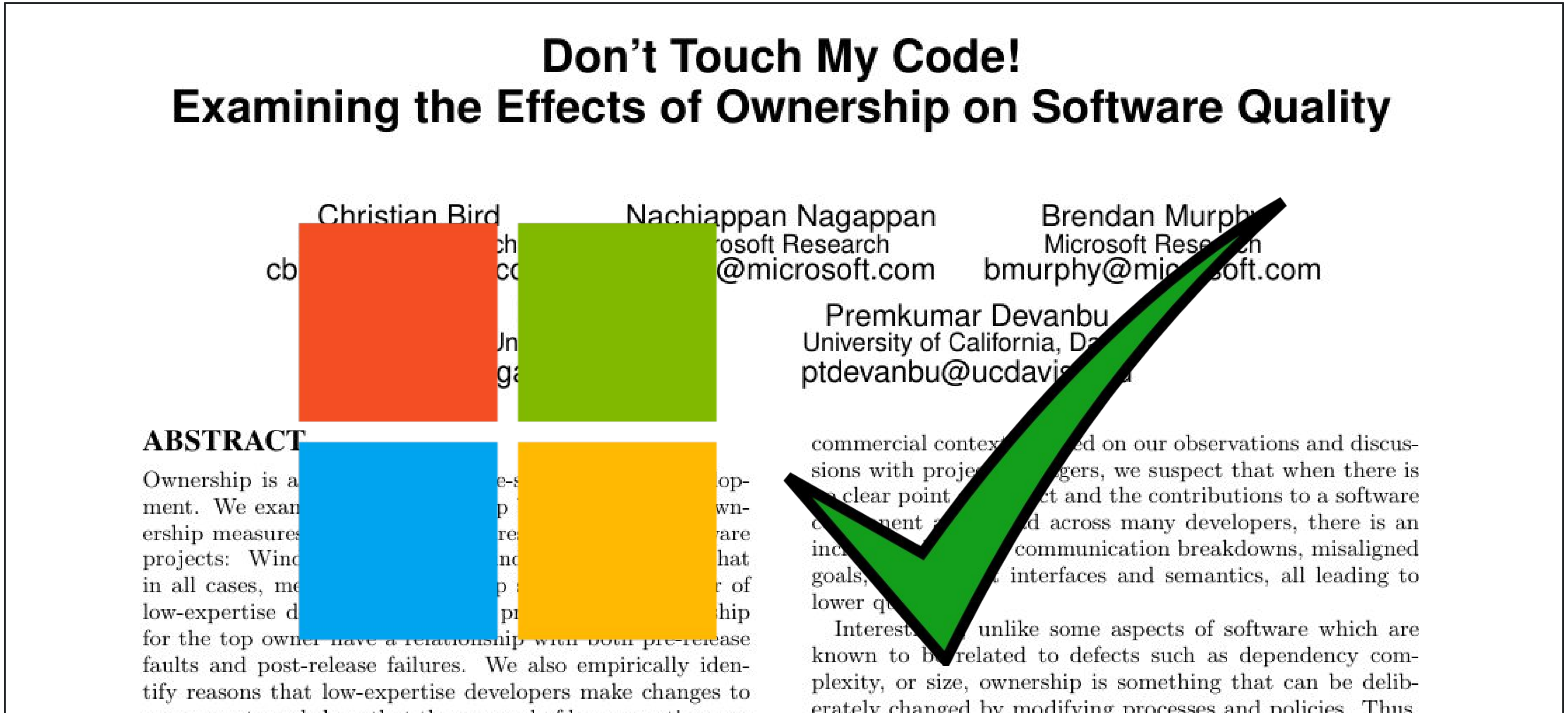
### ABSTRACT

Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to

commercial contexts. Based on our observations and discussions with project managers, we suspect that when there is no clear point of contact and the contributions to a software component are spread across many developers, there is an increased chance of communication breakdowns, misaligned goals, inconsistent interfaces and semantics, all leading to lower quality.

Interestingly, unlike some aspects of software which are known to be related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus

# [1] - Ownership and n° of defects



## Don't Touch My Code! Examining the Effects of Ownership on Software Quality

Christian Bird
cb

Nachiappan Nagappan
rosoft Research
@microsoft.com

Brendan Murph
Microsoft Rese ch
bmurphy@mi soft.com

Premkumar Devanbu
University of California, Da
ptdevanbu@ucdavi

### ABSTRACT

Ownership is a e-s op- ment. We exam p wn- ership measures re are projects: Wind nc hat in all cases, me p r of low-expertise d pr hip for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to

commercial contex ed on our observations and discussions with proje gers, we suspect that when there is clear point t and the contributions to a software c nent d across many developers, there is an inc communication breakdowns, misaligned goals, interfaces and semantics, all leading to lower q

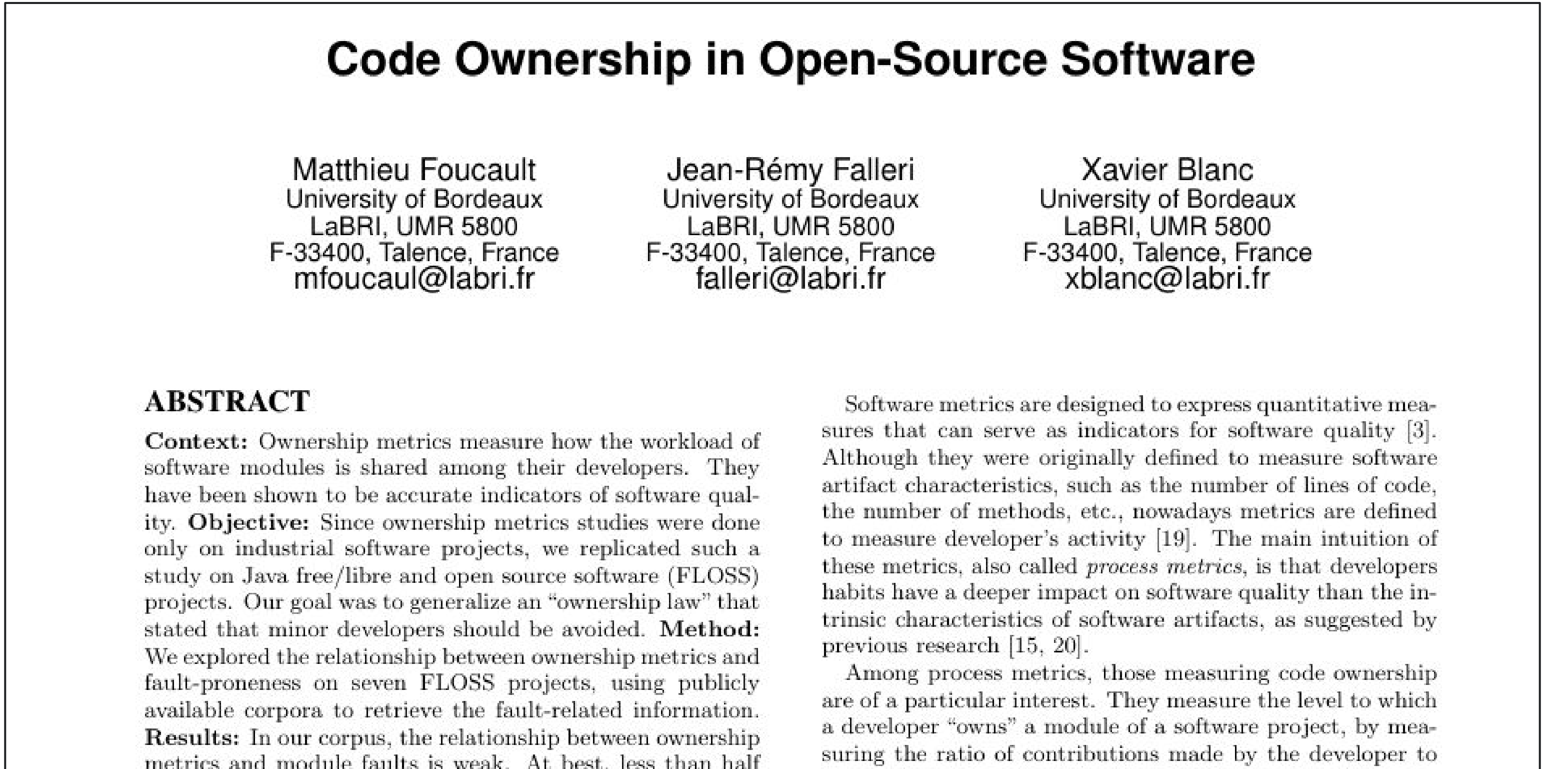
Interest unlike some aspects of software which are known to b related to defects such as dependency complexity, or size, ownership is something that can be deliberately changed by modifying processes and policies. Thus

# [1] - Ownership metrics

- **Ownership**: proportion of ownership of the highest contributor;
- **Total**: total number of contributors;
- **Minor**: number of contributors with proportion of own, **below 5%**
- **Major**: number of contributors with proportion of own, **above 5%**

Proportion in terms of **commit count**

# [2] - Replication on OSS



## 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

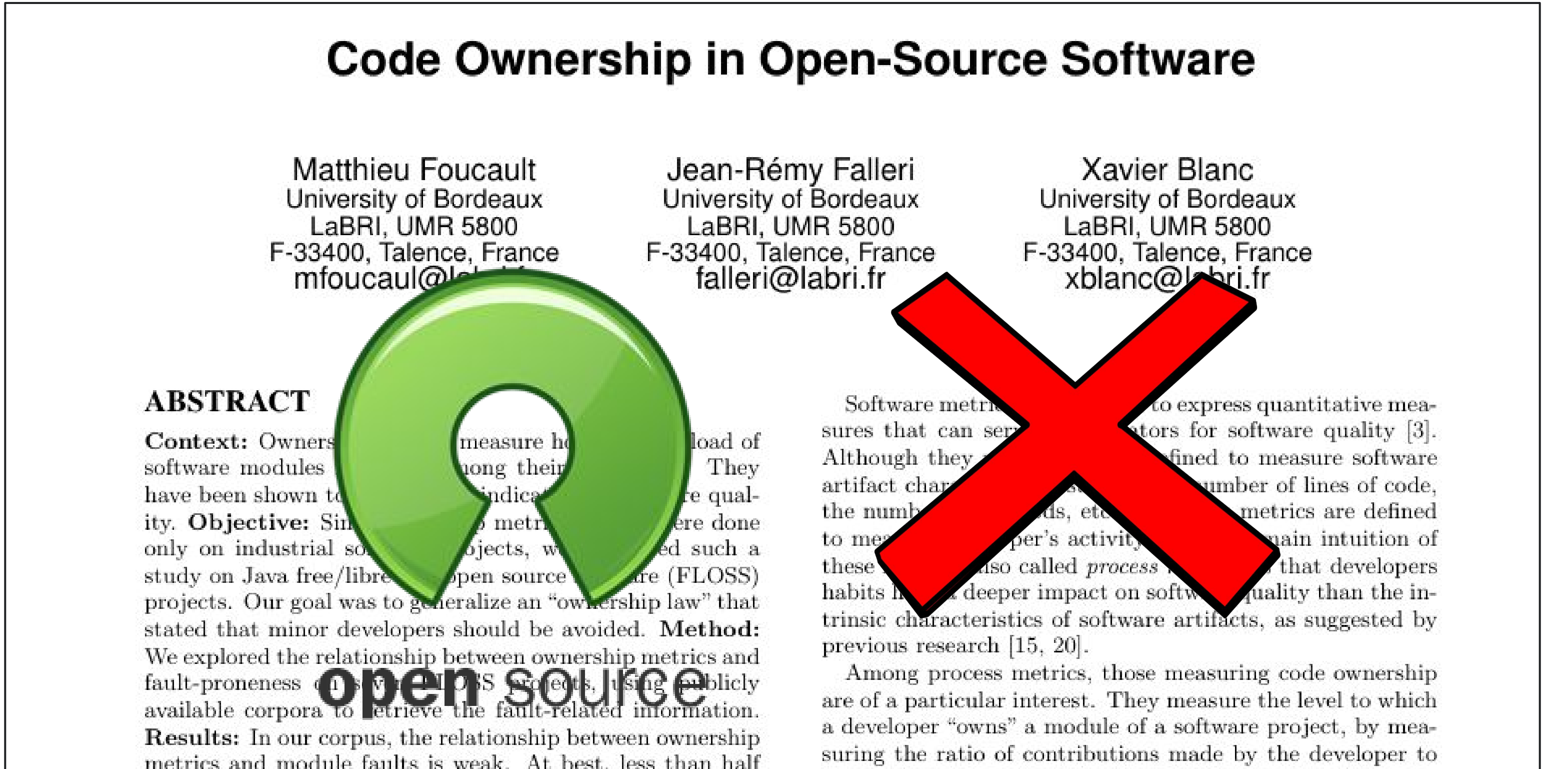
### ABSTRACT

**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:** Since ownership metrics studies were done only on industrial software projects, we replicated such a study on Java free/libre and open source software (FLOSS) projects. Our goal was to generalize an "ownership law" that stated that minor developers should be avoided. **Method:** We explored the relationship between ownership metrics and fault-proneness on seven FLOSS projects, using publicly available corpora to retrieve the fault-related information. **Results:** In our corpus, the relationship between ownership metrics and module faults is weak. At best, less than half

Software metrics are designed to express 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, the number of methods, etc., nowadays metrics are defined to measure developer's activity [19]. The main intuition of these metrics, also called *process metrics*, is that developers habits have a deeper impact on software quality than the intrinsic characteristics of software artifacts, as suggested by previous research [15, 20].

Among process metrics, those measuring code ownership are of a particular interest. They measure the level to which a developer "owns" a module of a software project, by measuring the ratio of contributions made by the developer to

# [2] - Replication on OSS



## Code Ownership in Open-Source Software

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

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@l bri.fr

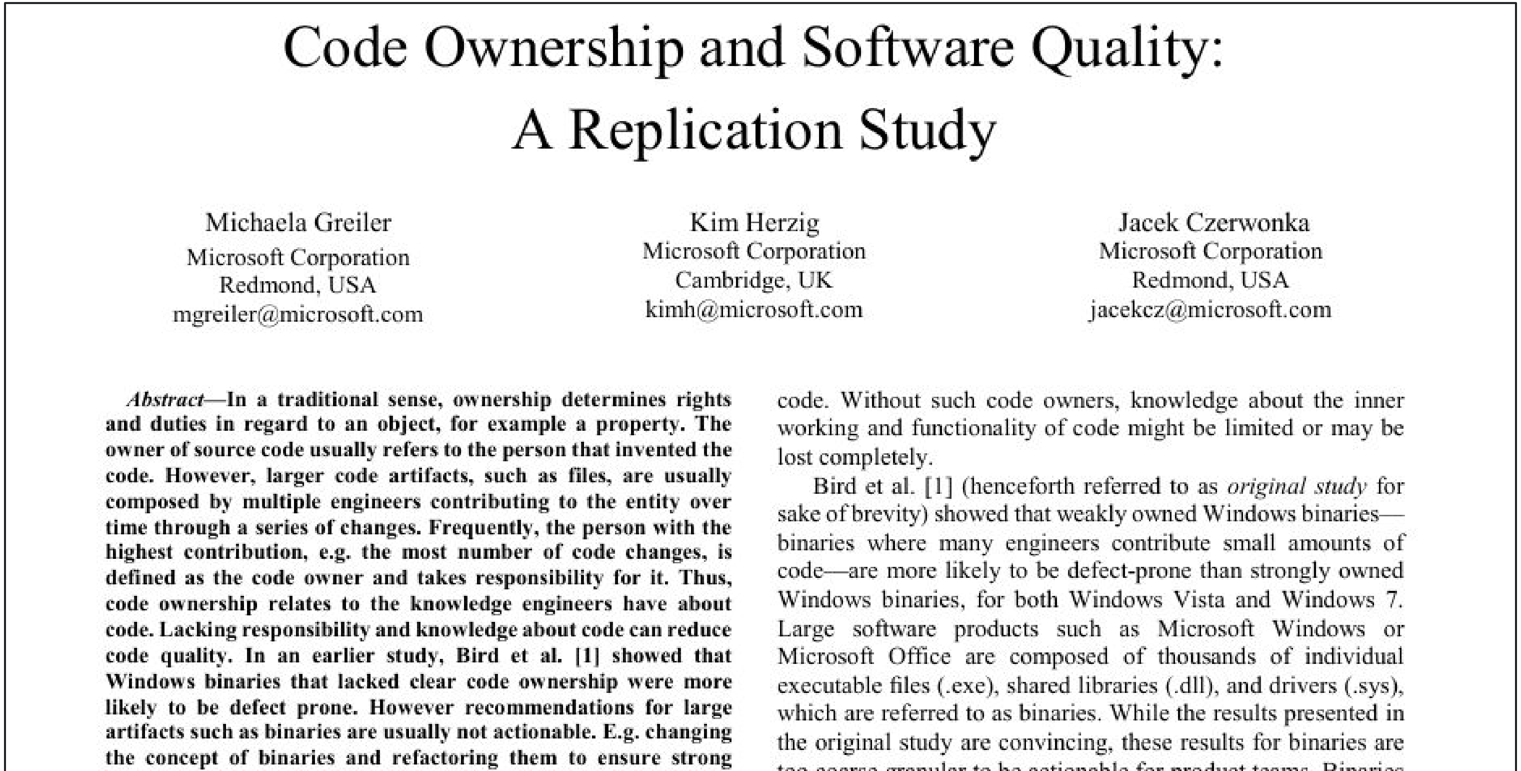
### ABSTRACT

**Context:** Owners measure h load of software modules ong their They have been shown t indica e quality. **Objective:** Si metri ere done only on industrial so ojects, w ed such a study on Java free/libre pen source re (FLOSS) projects. Our goal was to generalize an "ownership law" that stated that minor developers should be avoided. **Method:** We explored the relationship between ownership metrics and fault-proneness o s FLOSS projects, using publicly available corpora to etrieve the fault-related information. **Results:** In our corpus, the relationship between ownership metrics and module faults is weak. At best, less than half

Software metri to express quantitative measures that can ser tors for software quality [3]. Although they fined to measure software artifact char umber of lines of code, the numb ds, etc metrics are defined to me per's activity ain intuition of these lso called *process* that developers habits h a deeper impact on softw quality than the intrinsic characteristics of software artifacts, as suggested by previous research [15, 20].

Among process metrics, those measuring code ownership are of a particular interest. They measure the level to which a developer "owns" a module of a software project, by measuring the ratio of contributions made by the developer to

# [3] - Replication - classification of defects



## Code Ownership and Software Quality: A Replication Study

Michaela Greiler
Microsoft Corporation
Redmond, USA
mgreiler@microsoft.com

Kim Herzig
Microsoft Corporation
Cambridge, UK
kimh@microsoft.com

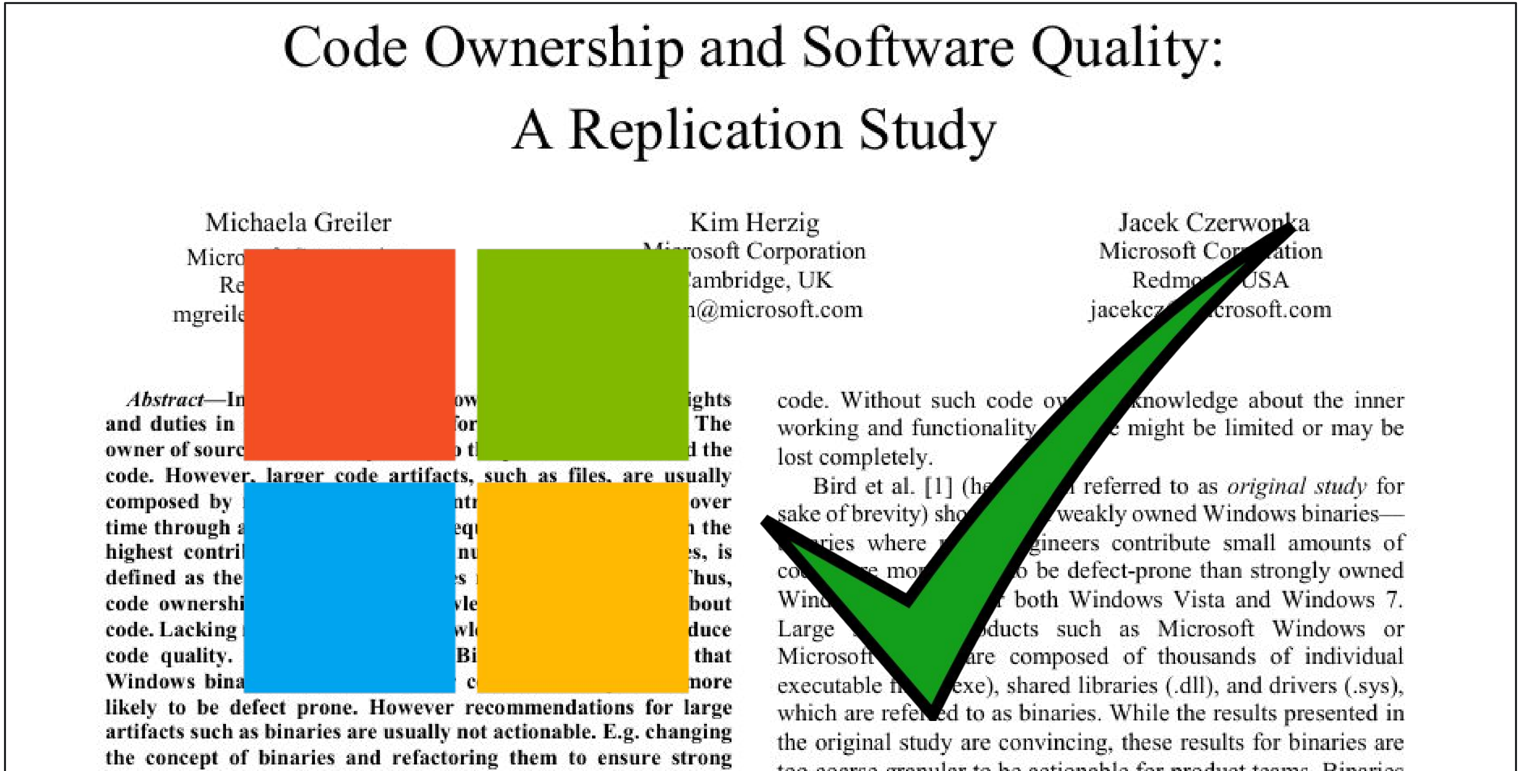
Jacek Czerwonka
Microsoft Corporation
Redmond, USA
jacekcz@microsoft.com

***Abstract*—In a traditional sense, ownership determines rights and duties in regard to an object, for example a property. The owner of source code usually refers to the person that invented the code. However, larger code artifacts, such as files, are usually composed by multiple engineers contributing to the entity over time through a series of changes. Frequently, the person with the highest contribution, e.g. the most number of code changes, is defined as the code owner and takes responsibility for it. Thus, code ownership relates to the knowledge engineers have about code. Lacking responsibility and knowledge about code can reduce code quality. In an earlier study, Bird et al. [1] showed that Windows binaries that lacked clear code ownership were more likely to be defect prone. However recommendations for large artifacts such as binaries are usually not actionable. E.g. changing the concept of binaries and refactoring them to ensure strong**

code. Without such code owners, knowledge about the inner working and functionality of code might be limited or may be lost completely.

Bird et al. [1] (henceforth referred to as *original study* for sake of brevity) showed that weakly owned Windows binaries—binaries where many engineers contribute small amounts of code—are more likely to be defect-prone than strongly owned Windows binaries, for both Windows Vista and Windows 7. Large software products such as Microsoft Windows or Microsoft Office are composed of thousands of individual executable files (.exe), shared libraries (.dll), and drivers (.sys), which are referred to as binaries. While the results presented in the original study are convincing, these results for binaries are too coarse granular to be actionable for product teams. Binaries

# [3] - Replication - classification of defects



## Code Ownership and Software Quality: A Replication Study

Michaela Greiler
Micro
Re
mgreile

Kim Herzig
Microsoft Corporation
ambridge, UK
n@microsoft.com

Jacek Czerwonka
Microsoft Cor ation
Redm USA
jacekcz crosoft.com

*Abstract*—In ow ights
and duties in for The
owner of sourc o t d the
code. However, larger code artifacts, such as files, are usually
composed by ntr over
time through a equ n the
highest contri nu es, is
defined as the s Thus,
code ownershi wle bout
code. Lacking owl duce
code quality. Bi that
Windows bina c more
likely to be defect prone. However recommendations for large artifacts such as binaries are usually not actionable. E.g. changing the concept of binaries and refactoring them to ensure strong

code. Without such code ow knowledge about the inner
working and functionality e might be limited or may be
lost completely.

Bird et al. [1] (he referred to as *original study* for
sake of brevity) sho weakly owned Windows binaries—
aries where gineers contribute small amounts of
co e mo o be defect-prone than strongly owned
Wind r both Windows Vista and Windows 7.
Large oducts such as Microsoft Windows or
Microsoft are composed of thousands of individual
executable fi exe), shared libraries (.dll), and drivers (.sys),
which are refe ed to as binaries. While the results presented in
the original study are convincing, these results for binaries are

# Related work shortcomings → Motivation

**(1)**

Metrics computed on a **release** and correlated with defects introduced **before**:

- ➢ We want to solve this main problem;

**(2)**

No change in **ownership granularity**: only commit-based

- ➢ We try also line-based;

**(3)**

No extensive study on the **minor-major threshold**:

- ➢ We want to better understand effects due to its value;

# Proposed solution

# In a nutshell...

*We want to see if ownership metrics can be used to* ***classify*** *defective and non-defective* ***Java files****, checking how much they can improve a model built using classic metrics known to be effective.*

*To solve the problem of* ***WHEN*** *computing the metrics we will compute them* ***for every version of every file****.*

# Defective files ?

**Implicated files**

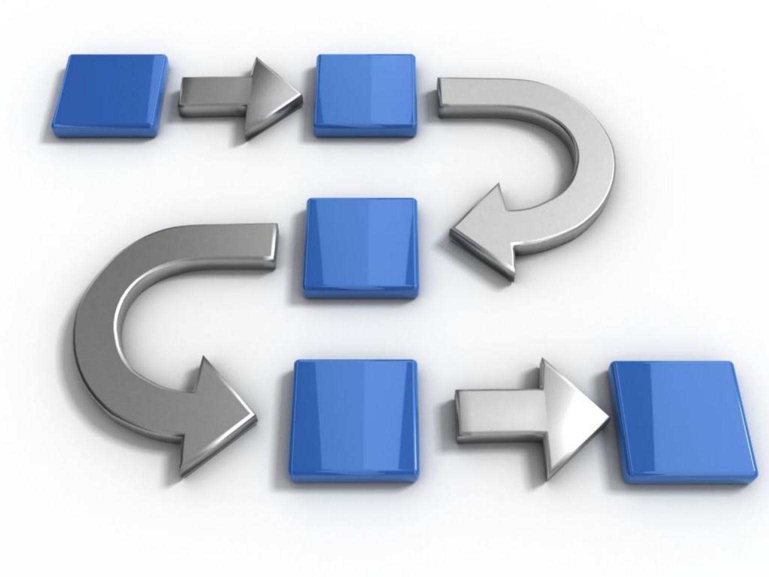
*File versions that **just introduced** some **code** that is **then deleted by a further bug fix** (i.e. implicated code)*

**Ownership, Experience and Defects:**

A Fine-Grained Study of Authorship

Foyzur Rahman and Premkumar Devanbu
Department of Computer Science
University of California, Davis, USA
{mfrahman,ptdevanbu}@ucdavis.edu

# Methodology



# Research questions

1. Are ownership metrics indicative for the presence of implicated code in source files for open-source software projects?

2. Can ownership metrics be used to build a classification model to identify implicated source files?

   2.1. For which level of granularity (line-based or commit-based) do ownership metrics give more accurate results when used to build the classification model?

   2.2. How does the value of the threshold, used to distinguish minor and major contributors, impacts on the accuracy of the classification model?

# Study subjects (Java projects)

Lucene
Apache Solr
JIRA
Apache
Maven
Apache Camel
mahout
ZOO
Apache Zookeeper

# Metrics

**Ownership metrics:**
Commit-based, line-based and memory-less (i.e. authorship)

for different minor-major thresholds: 5%, 10%, 20%, 30%, 40%

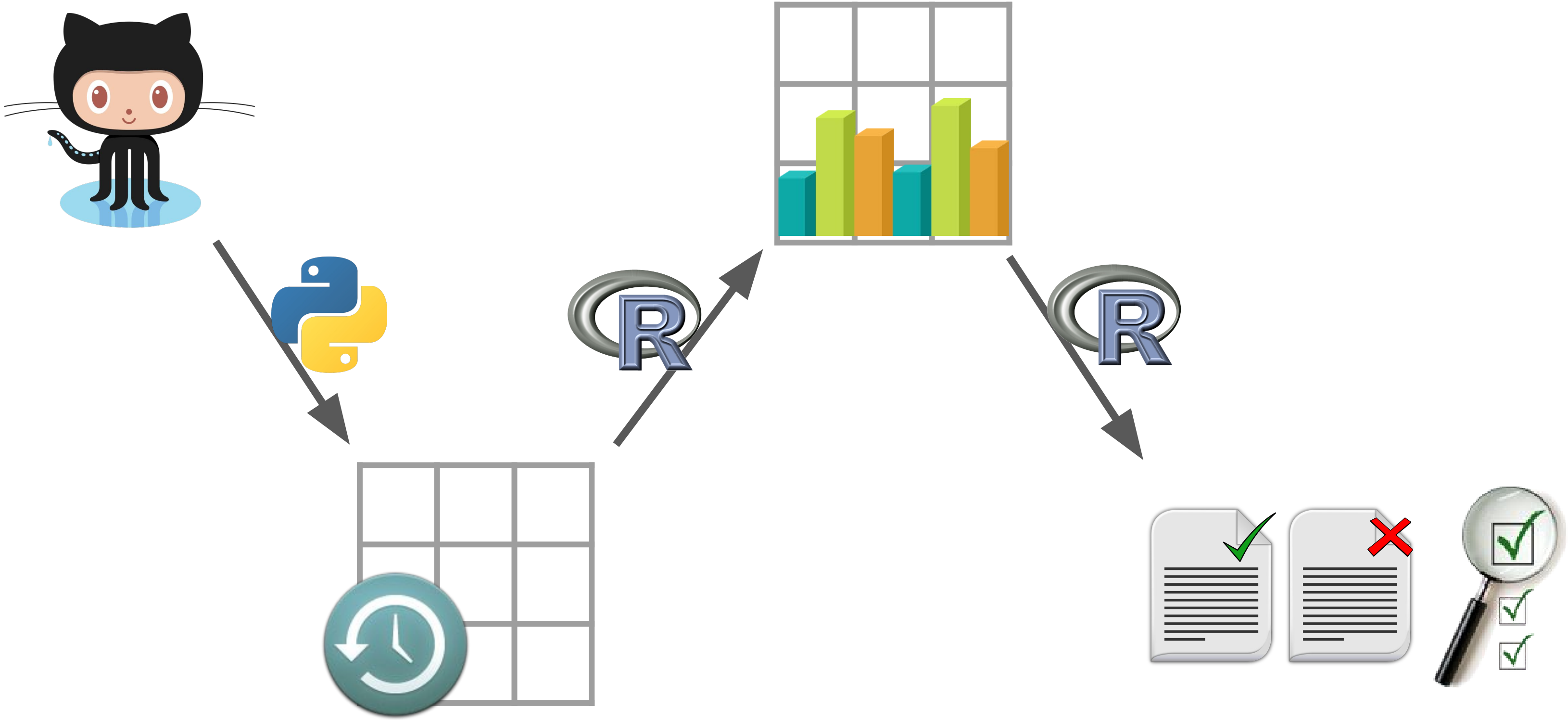
**Classic code metrics:**
File size, previous number of defects and comment-to-code ratio
**(known to be effective)**

**Ownership, Experience and Defects:**

A Fine-Grained Study of Authorship

Foyzur Rahman and Premkumar Devanbu
Department of Computer Science
University of California, Davis, USA

# Steps



# Evaluation & Results

# Experiment 1: Ownership and granularity

**Goal:**

*Determine if the ownership metrics can improve the accuracy of the classifier that uses only the classic metrics and if the granularity (line- or commit-based) has any impact on the accuracy.*

**Experiment**:

1. **Random Forest** classifier for every project using **different groups of metrics**;
2. Determine its **performance** using 10 fold cross-validation (CV);
3. **Compare** the performance of the groups **against the classic metrics**.
4. Determine if results are **statistically significant (paired t-test)**.

# Experiment 1 - Results (I)

**Lucene-Solr**

| **Features** | **Error Rate with 5% threshold** | **Improvement over CLASSIC** | **Precision** | **Recall** |
|---|---|---|---|---|
| Classic | 32.59% | 0.00% | 67.85% | 67.26% |
| Classic + Commit based | 30.38% | 6.78% | 67.21% | 70.62% |
| Classic + Deleted | 29.69% | 8.88% | 70.09% | 70.40% |
| Classic + Added | 29.49% | 9.51% | 69.40% | 70.98% |
| Classic + Line authorship | 29.46% | 9.59% | 74.29% | 69.11% |
| Classic + Line based | 26.12% | 19.86% | 80.41% | 71.13% |
| All metrics | 26.34% | 19.16% | 80.50% | 70.81% |

# Experiment 1 - Results (I)

**Lucene-Solr**

| **Features** | **Error Rate with 5% threshold** | **Improvement over CLASSIC** | **Precision** | **Recall** |
|---|---|---|---|---|
| Classic | 32.59% | 0.00% | 67.85% | 67.26% |
| Classic + Commit based | 30.38% | **6.78%** | 67.21% | 70.62% |
| Classic + Deleted | 29.69% | **8.88%** | 70.09% | 70.40% |
| Classic + Added | 29.49% | **9.51%** | 69.40% | 70.98% |
| Classic + Line authorship | 29.46% | **9.59%** | 74.29% | 69.11% |
| Classic + Line based | 26.12% | **19.86%** | 80.41% | 71.13% |
| All metrics | 26.34% | **19.16%** | 80.50% | 70.81% |

# Experiment 1 - Results (I)

**Lucene-Solr**

| Features | Error Rate with 5% threshold | Improvement over CLASSIC | Precision | Recall |
|---|---|---|---|---|
| Classic | 32.59% | 0.00% | 67.85% | 67.26% |
| **Classic + Commit based** | **30.38%** | **6.78%** | **67.21%** | **70.62%** |
| Classic + Deleted | 29.69% | 8.88% | 70.09% | 70.40% |
| Classic + Added | 29.49% | 9.51% | 69.40% | 70.98% |
| Classic + Line authorship | 29.46% | 9.59% | 74.29% | 69.11% |
| **Classic + Line based** | **26.12%** | **19.86%** | **80.41%** | **71.13%** |
| All metrics | 26.34% | 19.16% | 80.50% | 70.81% |

# Experiment 1 - Results (II)

| | Classic | Commit based | Line based | All metrics | Improvement (all metrics with FS) | Improvement (Line based) |
|---|---|---|---|---|---|---|
| **lucene-solr** | 32.59% | 30.38% | 26.12% | 26.34% | 19.16% | 19.86% |
| **mahout** | 31.68% | 29.44% | 25.42% | 29.33% | 7.43% | 19.77% |
| **Camel** | 29.36% | 25.16% | 19.38% | 19.25% | 34.43% | 34.00% |
| **Maven** | 25.51% | 23.19% | 19.27% | 19.09% | 25.15% | 24.45% |
| **Zookeeper** | 26.94% | 22.35% | 20.67% | 20.92% | 22.37% | 23.27% |
| **Average** | **29.22%** | 26.10% | 22.17% | **22.99%** | **21.71%** | 24.27% |

# Experiment 2: Logistic regression

**Goal**:

*Determine which metrics are statistically most significant for the classification.*

**Experiment**:

1. Use Logistic Regression to build a classifier
2. Retrieve the statistically significance of every metric

# Experiment 2 - Results

| **METRICS** | **Lucene-solr** | **Camel** | **Mahout** | **Maven** | **Zookeeper** |
|---|---|---|---|---|---|
| file size | *** | *** | *** | *** | |
| previous implications | *** | *** | *** | *** | *** |
| comment to code ratio | . | *** | *** | *** | *** |
| major contributors | *** | *** | NA | *** | *** |
| commit ownership | | *** | * | *** | . |
| minor contributors | NA | NA | ** | NA | NA |
| total contributors | *** | *** | *** | *** | *** |
| total authors | *** | | | | *** |
| line authorship | *** | *** | *** | *** | *** |
| line ownership added | | *** | *** | *** | . |
| lines added minor contributors | | NA | | NA | NA |
| lines added major contributors | | ** | . | | ** |
| line ownership deleted | . | *** | *** | | . |
| lines deleted minor contributors | NA | NA | | NA | NA |
| lines deleted major contributors | * | | * | ** | |
| **ACCURACY** | **0.6325** | **0.65** | **0.6925** | **0.6425** | **0.59375** |

# Experiment 3: Minor-major thresholds

**Goal**:

*Determine the influence of  the threshold on the performance of the classifier.*

**Experiment**:

1. Create for every project a classifier for every minor-major threshold (0.05, 0.10, 0.20, 0.30, 0.40).
2. Determine the performance of the classifier using 10-fold CV.
3. Determine if there is any statistically significant difference between the performance of the classifiers using the *ANOVA* test.

# Experiment 3 - Results

| Projects | OOB (0.05) | OOB (0.10) | OOB (0.20) | OOB (0.30) | OOB (0.40) |
|---|---|---|---|---|---|
| Lucene-solr | 44.60% | 43.35% | 43.60% | 44.99% | 46.17% |
| Camel | 39.15% | 37.75% | 39.14% | 40.26% | 41.23% |
| Mahout | 41.37% | 39.89% | 39.67% | 43.63% | 42.83% |
| Maven | 38.25% | 38.89% | 36.66% | 40.81% | 42.00% |
| Zookeeper | 33.44% | 33.72% | 34.82% | 36.93% | 34.73% |

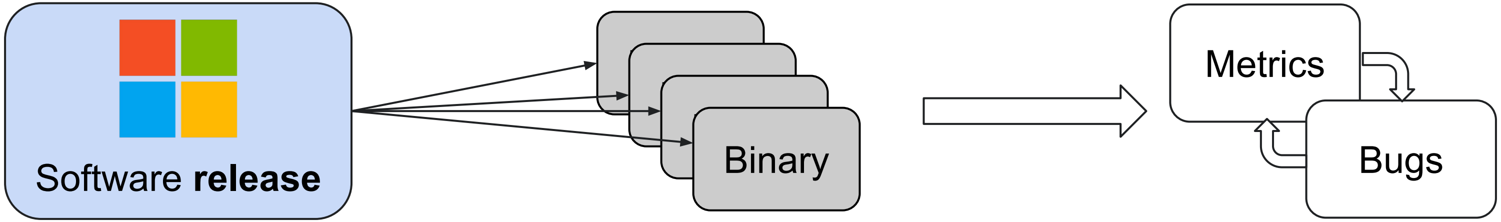
***No statistically significant difference between thresholds!***

# Conclusion

# Conclusion

1. Ownership metrics **are indeed indicative for implicated code in OSS**, but are also **highly project dependent → Total and Line authorship always significant as [2] and [4] stated.**
2. Different **minor-major threshold** don’t show any significant improvement for the classifiers;
3. **Line-based granularity performs** on average 15% **better** than commit-based metrics, when including memory-less ownership;
4. The constructed classifier
   - has an **out-of-bag error rate of 23%;**
   - and performs on average **22% better than just the classic metrics**.

# [1]



- **Ownership**: proportion of ownership [illegible] the highest contributor;
- **Total**: total number of contributors
- **Minor**: number of contributors [illegible] proportion of own, **below 5%**
- **Major**: number of contributo[illegible]ith proportion of own, **above 5%**

Proportio[illegible]ms of **commit count**

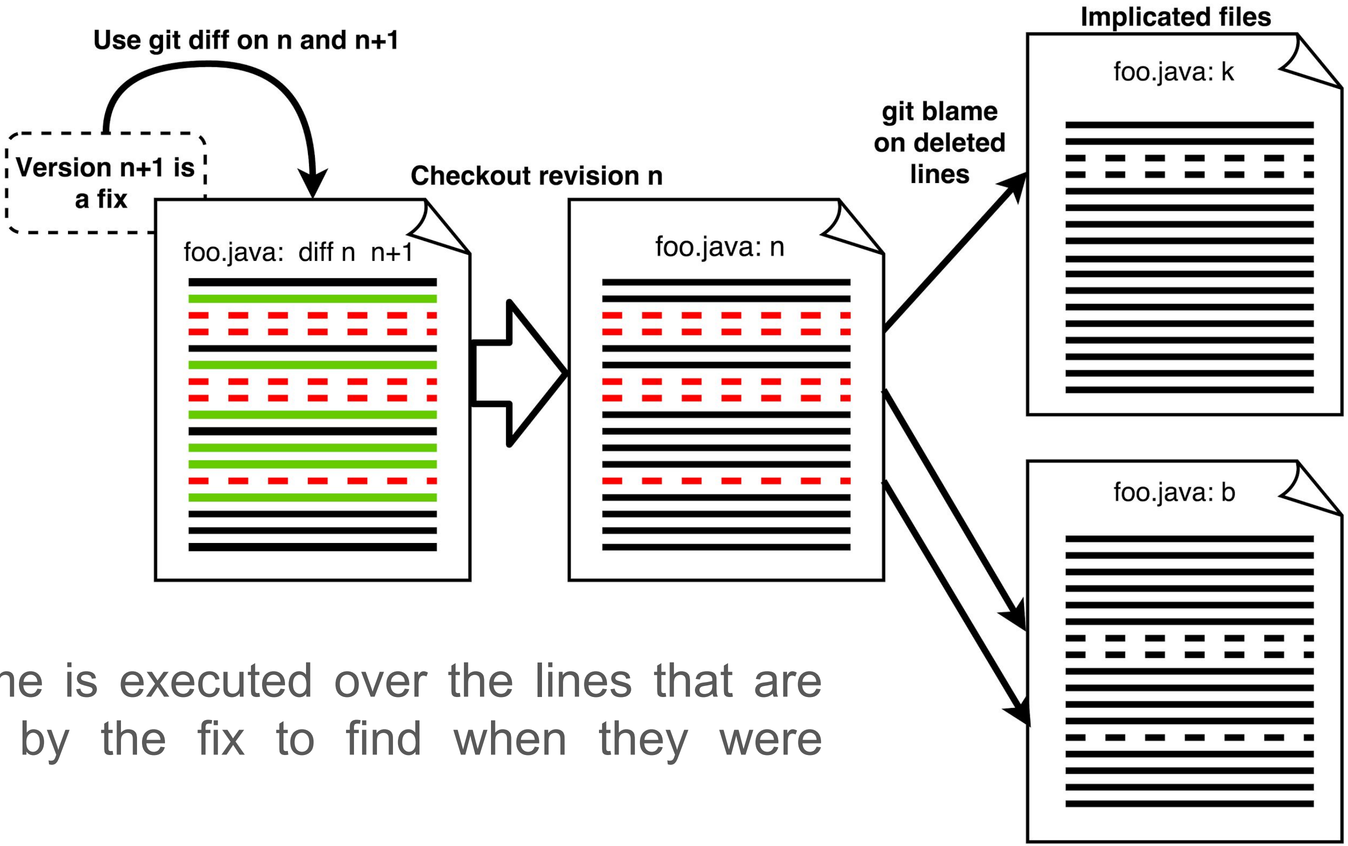
# Steps

1. Build a dataset that captures **the whole history** of development in terms of **code changes** (commit, lines), **authorship** and **bugs**;

2. Extract from that dataset a dataset with the **metrics computed for every version of every file** and the **column that says if the version is implicated or not**;

3. Use the metrics dataset to **build the models (Random Forest and Logistic Regression)** testing different sets of features;

# [4] → Implicated code extraction



Git blame is executed over the lines that are deleted by the fix to find when they were added.

# Ownership vs authorship

**Ownership**

Ownership metrics consider the history of the development (over a certain time period).

**e.g.:** number of lines added to F by C over all the considered history (all the versions of F that precede V), divided by the total number of lines added to F by all the contributors over all the considered history;

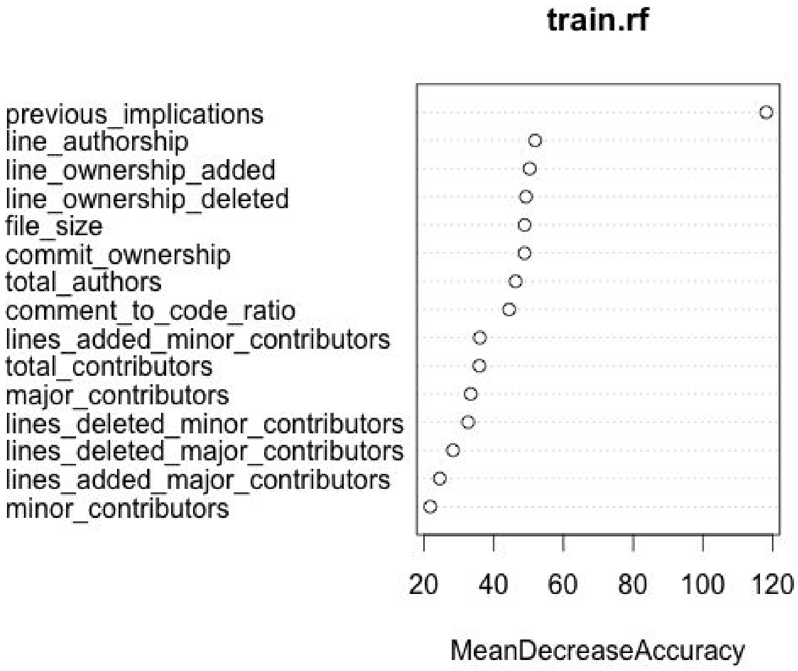
**Authorship (memoryless ownership)**

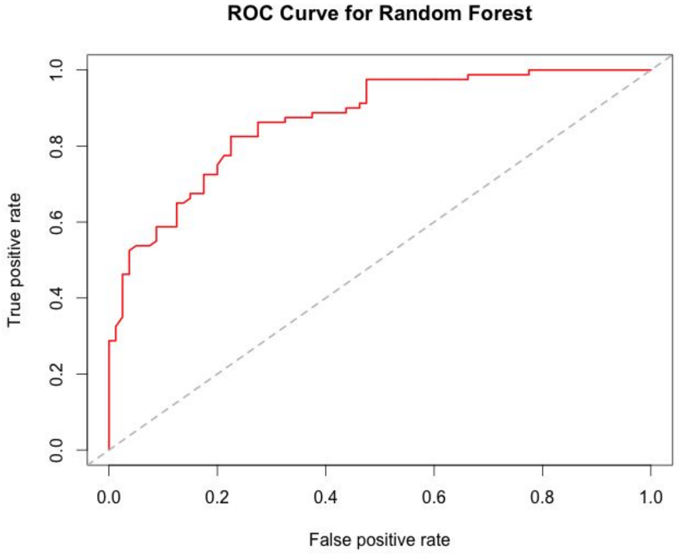
Authorship metrics consider only a specific revision of the software.

**e.g.:** number of lines that are actually in F in its version V and that are authored by C, divided by the size of F in the same version (in terms of lines);

# Experiment 1 - Results (III)

**Lucene-Solr**





# Future work

1. Repeat experiments on **more projects**, also with **languages other than java,** to see if the results can be more generalized.

2. See if **changing** the following parameters affects the results:
    - the **granularity of the considered artifacts** (e.g. folders or packages);
    - and the **history time period**.

# Conclusion