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 ownership would violate system architecture principles. A recent replication study by Foucault et al. [2] on open source software replicate the original results and lead to doubts about the general concept of ownership impacting code quality. In this paper, we replicated and extended the previous two ownership studies [1, 2] and reflect on their findings. Further, we define several new ownership metrics to investigate the dependency between ownership and code quality on file and directory level for 4 major Microsoft products. The results confirm the original findings by Bird et al. [1] that code ownership correlates with code quality. Using new and refined code ownership metrics we were able to classify source files that contained at least one bug with a median precision of 0.74 and a median recall of 0.38. On directory level, we achieve a precision of 0.76 and a recall of 0.60.**

***Index Terms*— Empirical software engineering, code ownership, software quality**

I. Introduction

In recent years, a number of studies investigated the impact of code ownership and organizational structure on software quality and reliability [1, 2, 3, 4, 5, 6, 7]. Code ownership—the number of engineers contributing to a source code artifact and the relative proportion of their contributions—determines collaboration networks among software developers. These networks and their underlying organizational structures influence the programming behavior of developers and their communication channels. As shown by previous studies code ownership and organizational structure impacts code quality. Code ownership and in particular the lack of ownership is likely to cause the lack of responsibility for code parts that an engineer does not own. Thus, code without a strong owner might have no champion who will take responsibility to maintain and test the 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 tend to comprise large numbers of individual code entities and are unlikely to be strongly owned by a single or even a few developers. However, distributed ownership for binaries does not imply that individual lower level code entities, e.g. source files, have the same weak ownership. A weakly owned binary can contain hundreds of code entities each owned by a single engineer. Further, recommendations for binaries are usually not actionable. Changing the concept of binaries and splitting existing binaries into smaller pieces for the sake of ensuring strong ownership would violate system architecture purposes and principles.

To make the results more actionable for engineers, we decided to look at two different granularity levels: we use the smallest logical and physical entity comprised in a binary, i.e., files as unit of investigation. However, bug distributions on file level can be very sparse leading to highly unbalanced data sets—data sets in which the number of source files that contained a code defect is very small, e.g. below 1%. The finer the level of granularity, the more unbalanced the data sets will become. Therefore, we decided to choose an intermediate level of granularity that lies between binaries and source files: *code directories*. Directories are logical groupings of files that often share some common properties. While we newly investigate the file level for Microsoft products, we replicate the study by Foucault et al. [2] who investigated open source systems on file level granularity previously. We check if results reported by Bird et al. [1] are valid for finer levels of granularity and generalize beyond Windows and whether results reported by Foucault et al. [2] on file level (i.e., no impact on quality) can be confirmed on Microsoft products.

More specifically, we make the following contributions:

- We widen the set of Microsoft products under investigation to four main Microsoft product: Office, Windows, Office365 and Exchange (Section III);
- We extended and refined the set of ownership metrics as defined by Bird et al. [1] in Section IV;
- We investigated how well the newly refined ownership metrics correlate with applied bug fixes on two actionable levels of granularity: code directories and source file level (Section III and Section V);
- We show that our ownership metrics can be used to predict software bugs with a median precision of 0.76 and a median recall of 0.60 on directory level (Section V);
- We show that the number of contributors and the percentage of edits of the lowest contributor seem to be the most important code ownership metrics when classifying defective code files and code directories (Section V);
- We provide a set of recommendations for managing ownership (Section VI.B).

II. Summary of the Original Study

In this section, we give a brief overview of the original study design [1], but refer the interested reader to the original study for more details. The original study investigated two releases of the Windows software system: Windows Vista and Windows 7. For both systems, the authors assessed whether their defined ownership metrics relate to pre-release and post-release defects.

In particular, the original study used the following terminology and metric definitions, which we closely follow:

- A *contributor* has made commits/software changes to the software component.
- A *software component* as a unit of development that has some core functionality.
- A *minor contributor* is a contributor who made less than a predefined threshold of the changes to a component.
- A *major contributor* is a contributor who made more than the predefined threshold of changes to a component.
- *Ownership* is defined as the proportion of commits a contributor made to a component relative to the total number of commits to that component.

Based on those definition, the authors define those metrics:

- *Minor*: the number of minor contributors
- *Major*: the number of major contributors
- *Total*: the number of total contributors
- *Ownership*: the percentage of changes of the contributor with the highest number of changes.

The authors used spearman rank correlation to check whether their ownership measurements relate to the number of pre-release and post-release code defects for each release.

In addition, the authors performed multiple linear regression to compare normal software measurements (such as size and complexity) with ownership measurements. The authors conclude that for all binaries the defined ownership metrics have a relation to pre- as well as post-release defects.

III. Experimental Setup

This section details our experimental setup. It includes our research question, descriptions of our research methodology and evaluation, and explains deviations to the original study.

A. Research Questions

To understand the impact of ownership measured on source file and directory level on code quality, we investigate the following research questions:

RQ1: Are ownership metrics indicative for code quality for other software systems than for Windows?

RQ2: For which levels of granularity (directory and source file) do ownership metrics correlate with code quality measured by the number of fixed bugs?

RQ3: Can ownership metrics be used to build classification models to identify defective directories and defective source files?

RQ4: What are the reasons for a lack of ownership?

B. Study Subjects

We investigated four major Microsoft products: Office, Office365, Exchange, and Windows.

We chose this set of products for a number of reasons. First, the set of products represents products of different nature. While Office and Windows are classical box products that are shipped to customers, Office365 and Exchange represent service products[1]. Second, each product, even within a group of products, was developed using different development methodologies and cultures, e.g. different branching structures, release cycles, componentization concepts, etc. However, each product in itself is a complex system under active development with hundreds of engineers and major development efforts for each release.

For each of the investigated products we analyzed one to three release cycles, reaching from 3 years for Exchange to 1 year for Office. For Office, we analyzed two releases Office 15 (OF15) and Office 16 (OF16). For Exchange we analyzed the two major releases Exchange 2013 (Ex13) and Exchange 2015 (Ex15). Additionally, we analyzed one release of Windows (Win) and one release of Office365 (O365).

Each of these software systems comprises several millions lines of code, and during the respective release cycles several hundred thousand changes were applied.

C. Measuring Code Quality

For this study, we use data provided by a Microsoft internal mining tool called CodeMine [8]. CodeMine is a platform that comprises mined repository data for all major Microsoft products. Among these repositories are version control systems and bug databases which allow us to identify file changes from

[1] There exist many differences between box and service products. Most importantly, code defects for service products can be fixed within seconds by rolling back changes. Defects shipped to customers in box products require customer action and tend to be more expensive.

version control commits and to link these changes to bug reports. We measure code quality by counting the number of bug fixes that are linked to a code artifact, e.g. a source file. We assume, the higher the number of bug fixes, the lower the code quality of that artifact. The bug data we use represents all bugs fixed during the investigated release cycles (see Section III.B). Thus, bug data sets for a release N of a product P contains pre-release bug fixes for release N. Some of these bug fixes might be post-release fixes for a release N-1 or earlier. CodeMine parses commit message (product teams used keyword identifiers to mark bug references) of applied code changes to identify bug reports that were referenced by the applied code change. The identified bug reports are then associated with all files checked-in by this commit. For directories, we aggregate the number of distinct bugs associated to all source files.

Independent of the product under study, the majority of files have no bug fixes assigned. Only a very low number (less than 0.02%) of files are outliers with more bugs assigned. Thus, our data sets are highly unbalanced and the vast majority of code artifacts will be defect free. This is especially true on source file level, making it hard for any machine learner to learn from the few defect prone instances. This is one of the reasons why we are also investigating source directory level where the data sets will be less deranged, but still unbalanced.

D. Statistical Evaluation and Bug prediction

To understand the relations between our ownership metrics and the number of fixed bugs, we investigate correlations between the ownership metrics defined in Section IV and code quality measures defined in Section III.C. Further, we train and test *random forest* machine learners to build classification models to identify files and directories that are associated to at least one bug fix. Finally, we use metric importance analysis to estimate the predictive power of the different metrics.

Our definition of code ownership will be based on code changes (see Section IV). Artifacts changed only once will be strongly owned. These instances are not helpful to establish a model using ownership values to decide whether a code artifact is defective, we ran two independent sets of experiments. One in which we consider all code artifacts and one for which we remove code artifacts that were changed only once. All statistical experiments were conducted using the R statistical software[2].

Correlations

Correlation values presented in this paper are—as in the original study—Spearman rank correlation values suitable for non-normalized data. Correlation values lie between -1 and 1 and describes how well the dependency between two metrics can be described using a monotonic function. A correlation value of 1 or -1 occurs when one metrics is a perfect monotone function of the respectively other measurement. Nearly all code metrics highly correlate with size measures; e.g. the higher the number of files in source directory, the higher the number of weakly owned files. Thus, we normalized metrics by corresponding size measurement. For more details on the size measure used for normalization, please see the metrics description in TABLE I.

All metric correlations have been checked for statistical significance by performing a Spearman correlation test and a Mann-Whitney U test. All reported values are statistically significant showing p-values below 0.01. As for any other study that uses statistical models to investigate relational phenomena please remember that correlation does not imply causality.

Classification Models

Further, we use *random forest* machine learners[3] [9] to train and test classification models to identify defective source files and directories. Analogue to the original study, classification models trained and tested in this study distinguish code artifacts that were fixed at least once from those that were not fixed.

For each product, we split the overall data set of files and directories into two subsets. One subset containing two third of the original data points for training purposes, the remaining data for testing purposes. To split the data into representative samples, we used a stratified repeated holdout setup—sampling preserves the proportion of positive and negative instances in the original data in both training and testing sets. We further remove metrics that show correlation values of 0.9 or above to any other metrics. Using principal component analysis (PCA) we ensure that all feature vectors used to train the classification model are orthogonal to each other. Thus, all feature vectors used to train the classification models are independent. Each dataset is sampled 10 times (10 cross-fold). We report the mean precision, recall, and f-measure values.

Metric Importance

To estimate the predictive power of individual ownership measurements, we used the *filterVarImpl* function of the caret package [10] for R. The function conducts "a series of cutoffs […] to the predictor data to predict the class. The sensitivity and specificity are computed for each cutoff and the ROC curve is computed. The trapezoidal rule is used to compute the area under the ROC curve. This area is used as the measure of variable importance" [10]. Please note that these metric importance measures may not match the spearman rank correlation results. While the rank correlation considers the exact order of all entities, classification models simply separate entities into categories. The suitability of a metric to solve either problem may be different.

E. Qualitative Evaluation

We collaborated with engineers from all investigated products to ensure that we understand their development processes (bug assignments, release cycles, methodologies and style of collaboration), which is crucial for a sound data analysis.

We further choose Microsoft Office as the product we engage with to complement our statistical evaluations with interviews and close interaction on the research results. Office seemed of particular interest for this study as they specifically encourage people to move within the organization leading to "collective" code ownership.

[2] http://www.r-project.org/

[3] Random forests are ensembles of decision tree classifiers that grow multiple decision trees each voting for the class an instance to be classified.

With Office, we had several feedback loops over the course of 10 month. The close collaboration with product teams comprised one-on-one discussions, presentations for several different teams, and one-on-one interviews with 6 engineers familiar with the source code. Insights gathered during those sessions were crucial to understand the different reasons for weak code ownership, their implications and the actionable countermeasures.

Participants were chosen based on their knowledge about particular files and sub directories in components (such as Excel, Word, PowerPoint, etc.), time availability and willingness of participation. Most of the engineers interviewed where Software Developers or Software Development Leads. Also a Tester and a Product Manger have participated in discussions and meetings. In addition to the 6 Office team members, 3 more engineers of other teams were involved in the discussions.

IV. Measuring Ownership

There exist multiple ways to measure ownership, especially for source code. In our particular case, we were interested in a measurement that reflects the notion of responsibility—how many engineers are currently contributing to the code and does a main contributor exist? The assumption is that main contributors show responsibility for their code and act as a human quality guard, e.g. reviewing code changes and writing tests to ensure high quality. Modifying code that has no such guard may risk quality issues.

Analogue to the original study, we base our ownership metrics on check-ins to code repositories, i.e., file changes that have been committed to the code repository. We also treat one check-in as one change to the files attached—irrespectively of the size of the change, i.e., how many line of code have changed. Lines of code (churn) is in general a difficult metric as changing one line in one file, might represent a much more complex task than adding a new method comprising several lines of code. To make a change to a file, the engineer has to be knowledgeable about the file irrespectively of the number of lines changed. We only consider changes to files that comprise source code or product configuration information, excluding non-code files such as images or build related configurations. Please note that we also ignore code changes that apply bug fixes for ownership measurements. Bug fixes relate to corrective rather than inventive maintenance tasks and thus have only limited value when measuring code ownership. Nevertheless, we replicated the whole analysis with the complete set of changes. The results are very similar to the outcomes reported in this study. No significant changes could be observed.

In the original study, *components* were large Windows binaries. In our study, we deviate in the definition of software component, as we will look either at *source code files* or at *source directories* that contain source code files as a unit of development. We group all source files that have the same enclosing system path name, e.g. "c:\source\directory\name", into the same group of source files, referred to as *directory*. Please note that this definition is not recursive. Thus, code files located in a directory $D_{level\ n}$ that is a sub-directory of another directory $D_{level\ n-1}$ belong only to $D_{level\ n}$ but not to $D_{level\ n-1}$. On file level, we made a differentiation between files that have only been edited once in the release, and files that have been edited at least twice. We did that because files that have been edited by only one engineer can per default be only strongly owned. These files will not help us to establish a connection between ownership and code quality; in fact, these files may influence correlation values. As discussed in Section III.D, we ran two separate experiments, one set of experiments includes all files and one set ignores files changed only once. As we could not easily define a similar exclusion on directory level, we calculated correlations, precision and recall for all directories, regardless how often files have been edited.

Because the size of a software component differs from the original study, we also deviate from the original percentage of the contributions considered for minor and major contributors. Originally, the threshold for minor and major contributors was set to *5%*. A developer that applied more than 5% of all changes during the monitored time window was considered a major contributor. Source files tend to have much fewer contributors. Thus, the percentage of contributions made by one engineer also tends to be differently distributed as for large binaries, a fact that Foucault et al. [2] ignored in their study leaving the threshold at the original 5% as defined by Bird et al. [1]. After analyzing distribution values, we set the threshold for minor and major contributors to 50%. This means that a contributor that commits 50% or more of the changes is seen as *major contributor*, and a contributor with less than 50% changes is a *minor contributor*.

In addition to the metrics originally defined by Bird et al. [1] and briefly discussed in Section II, we added few new ownership metrics and also metrics that reflect the organizational structure of a team or division, as detailed in the next section.

A. Metrics

We divide our metrics in individual and organizational ownership metrics for both levels of granularity: source files and directories. All metrics are summarized in TABLE I.

Individual Ownership

The set of individual ownership metrics addresses the direct involvement of engineers by measuring their commits to the code base. On file level, we look at the total number of *contributors*, the percentage of commits for the contributor with the highest number of commits (*ownership*), and the number of contributors with less than 50% of ownership (*minors*), and less than 20% of ownership (*minimals*). Those metrics follow very closely the originally defined metrics.

For directory level, we aggregate the originally defined and slightly modified metrics, by aggregating file based ownership metrics and by normalizing those using appropriate size measurements. Note that given our definition of file level metrics, those do not need to be normalized as they are based entirely on the number of edits and not on the length of the code. Details on metric formulas can be found in TABLE I.

In particular, for directories we look at the average of the *ownership* values (*avgownership*), the percentage of commits of the highest contributor (*ownershipdir*), and the average number of distinct contributors (*avgcontributors*). Additionally, we measure the average number of minor contributors (*avgminors*),

TABLE I. OWNERSHIP MEASUREMENTS USED IN THIS STUDY.

Metric name	Description
	Individual Ownership for Files
ownership	Proportion of commits for the highest contributor.
minors	No. of contributor with ownership of less than 50%.
minimals	No. of contributors with ownership of less than 20%.
contributors	No. of contributors.
	Individual Ownership for Directories
avgownership	AVG *ownership* values for all files in that directory: *(sum of file ownership) / (#files).*
ownershipdir	Pct. of commits of the highest contributor considering all files in a directory: *(sum of all commits of main contributor) / (#all commits)*
minownerdir	Pct. of commits of the lowest contributor considering all files in a directory: *(sum of all commits of the lowest contributor)/(#all commits)*
avgcontributors	AVG of distinct *contributors* among all files in a directory: *(sum of distinct contributors per directory)/(#files)*
pcminors	Pct. of contributors among all contributors with **less than 50%** commits among all files in a directory: *(sum of distinct minors)/(#contributors)*
pcminimals	Pct. of contributors among all contributors with **less than 20%** commits among all files in a directory: *(sum of distinct minimals)/(#contributors)*
pcmajors	Pct. of contributors among all contributors with **more than or 50%** commits among all files in a directory: *(sum of distinct contributors with more than 50% changes)/(#contributors)*
avgminimals	AVG *minimals* in a directory: *(sum of minimals per file/#files)*
avgminors	AVG *minors* in a directory: *(sum of minors per file/#files)*
minownedfile	The *ownership* value of the file with the lowest ownership value.
weakowneds	No. of files in a directory that have an ownership value of less than 50%: *(num of files with < 50% ownership)/#files*
	Organizational Ownership for Files
manager3 or manager4 or manager5	No. of distinct managers contributors of a file have on level 3, or level 4 or level 5 counted from the distance to the CEO.
	Organizational Ownership for Directories
manager3dir or manager4dir or manager5dir	AVG no. of distinct managers contributors of all files in a directory have on level 3, or level 4 or level 5 counted from the distance to the CEO: *(sum of distinct managers per file/#files)*

the percentage of distinct minor (*pcminors*), minimal (*pcminimals*), and major (*pcmajor*) contributors. Apart from those direct derived metrics, we also defined additional directory specific measures. In the original study, Bird et al. revealed that the more minor contributors a binary has the more vulnerable it is for defects. We feel that in addition to the original minor metrics counting the number of contributors, a more fine-grained metric covering the percentage of commits of the lowest contributor (*minownerdir*) might be a good indicator of ownership. The smaller the percentage, the more likely this contributor has limited expertise in this area and therefore introduces more defects. Similar, we use the file with the lowest *ownership* value per directory (*minownedfile*)[4], as we feel this value could give additional predictive power in addition to the average values. Finally, we measure the percentage of weak owned files, i.e., files that do not have a strong contributor making more than half of the commits (*weakowneds*).

Organizational Ownership

This metric set abstracts from the individual engineers, and looks at organizational structures, i.e., management hierarchies to determine ownership. The rationale behind organizational structures as code ownership metrics is twofold: First, organizational structures not only determine which engineers work together, but also influence their programming behavior and communication channels and thus also the quality of source code. In addition, collective code ownership might decrease the ownership of individual engineers per artifact, but a strong ownership might still be visible when measured at team level.

The organizational structure of development teams can be represented as an organizational tree. Each node of the tree corresponds to a person. Manager nodes have children representing persons directly reporting to the manger. We can translate the levels in a tree into management levels. The root node is the CEO of the company, i.e., manager level zero. Managers directly reporting to her are level one managers, etc. A more detailed discussion is presented by Nagappan et al. [4].

For Microsoft, we can infer that managers represent at level 3 company divisions, at level 4 broader product teams, and at level 5 smaller sub teams within product teams. The organizational tree allows us to measure how many different teams and even divisions contribute to source code files and directories. We interpret the organizational tree as a description of official communication paths and responsibilities. Two engineers reporting to the same manager are likely to have daily personal contact compared to engineers having no common direct manager. Therefore, we derive the organizational metrics by counting all manager at a certain level instead of the individual engineers that perform the commits. For example, when four engineers commit to a single file, but only three report to the same manager on level 5, then our organizational metric would indicate 2 level 5 managers, whereby our individual metric would count 4 contributors. The metrics are called *manager3*, *manager4* and *manager5*. Analogue to other directory metrics, we use the average number of managers associated to code file changes in that directory. We will refer to these metrics as *manager3dir, manager4dir*, and *manager5dir.*

V. RESULTS

In this section, we present the results of our experiments described in Section III.

A. Descriptive Statistics

This section details some of the ownership statistics we could observe among the four projects and their releases. The actual values can be found in TABLE II. In general, we see that most

[4] *Minownedfile* is referencing one file only and therefore does not require any size specific normalization.

TABLE II. DESCRIPTIVE STATISTICS OF OWNERSHIP METRICS. M REFERS TO THE MEDIAN, 10% TO THE HIGHEST VALUE FOR THE LOWEST 10% OF THE DATA, 90% FOR THE HIGHEST VALUE FOR 90% OF THE DATA, Q1 FOR THE FIRST AND Q3 FOR THE THIRD QUANTILE.

Description of measurement		OF15	OF16	Ex13	Ex15	O365	Win
On File Level							
No. of contributors per file	Q3	1	1	2	2	2	2
	90%	2	2	4	3	3	3
% ownership of main contributor	10%	.5	.5	.41	.43	.5	.46
	Q1	1	1	.53	.67	.81	.5
No. minors	Q3	0	0	1	0	0	0
	90%	1	1	3	3	2	3
On Directory Level							
No. of files in a directory	Q1	1	1	1	1	3	1
	M	2	2	2	2	6	3
	Q3	5	6	6	5	11	7
% ownership of main contributor	Q1	.71	.89	.5	.5	.5	.67
	M	1	1	.77	1	1	1
% ownership of min contributor	Q1	.5	.5	.09	.24	.29	.23
	M	1	1	.4	1	1	1
% of minors per directory	Q3	0	0	.89	.67	.5	0
	90%	0.90	0.67	1	1	1	.6

TABLE III. SPEARMAN CORRELATION BETWEEN OWNERSHIP METRICS AND BUG NUMBERS ON FILE LEVEL FOR FILES WITH CHANGES ≥1 AND ≥2.

Metric	OF15		OF16		Ex13		Ex15		O365		Win	
	≥1	≥2	≥1	≥2	≥1	≥2	≥1	≥2	≥1	≥2	≥1	≥2
ownership	-.30	-.23	-.26	-.27	-.37	-.36	-.32	-.14	-.48	-.42	-.25	-.25
minors	.34	.35	.26	.29	.41	.42	.33	.26	.50	.46	.34	.36
minimals	.30	.32	.33	.39	.38	.39	.31	.32	.42	.37	.36	.38
contributors	.31	.31	.27	.32	.40	.42	.34	.25	.49	.47	.29	.33
manager3	-.04	-.04	-.10	.16	.27	.27	.13	.03	.29	.35	.20	.24
manager4	.15	.13	-.09	.15	.27	.28	.10	.01	.29	.35	.15	.19
manager5	.15	.13	.12	.17	.27	.29	.19	.02	.31	.36	.22	.25

of the files have only one or two contributors in a given release, across all investigated products. Only 10% of files have more than 2 or 3 contributors. We also see that for the Office product family (OF15, OF16, and the service product O365) the average ownership is stronger than for Exchange and Windows. For the Office product family, almost 75% of all files are completely owned by one main contributor (i.e., almost 100% of the commits come from one contributor). For Windows and both Exchange releases, we see that at least 10% of all files are modified by 3 or more minor contributors.

For directories, we see that most directories only contain a very small number of files that churn in a given release. For OF15, OF16, Ex13, and Ex15 the median number of files is 2. For Win the median is 3, while O365 has a median of 6 files per directory. Even on a directory level, we can see a strong sense of ownership: the majority of directories is completely owned by one person. Ex13 is the exception for which the main contributors have an *ownership* of 77% for more for 50% of the directories. Even for the percentage of edits coming from the owner with the smallest number of edits (*minowner*) the relative number of edits is quite high. Only for the 25% percentile of the files, the *minowner* contributes less than 25%, whereby we see the smallest contributions for Ex13. For OF15 and OF16, the *minowner* makes up to 50% of the edits in the 25% percentile.

TABLE IV. SPEARMAN CORRELATION BETWEEN OWNERSHIP METRICS AND BUG NUMBERS ON DIRECTORY LEVEL.

Metric	OF15	OF16	Ex13	Ex15	O365	Win
avgcontributors	.16	.16	.17	.16	.25	.08
avgownership	-.48	-.52	-.53	-.46	-.68	-.47
ownershipdir	-.50	-.54	-.53	-.48	-.65	-.42
minownerdir	-.52	-.57	-.59	-.56	-.68	-.53
avgminor	.51	.52	.52	.45	.69	.51
avgminimals	.43	.45	.53	.47	.65	.53
pcminors	.55	.58	.56	.51	.69	.55
pcminimals	.51	.57	.55	.56	.69	.59
pcmajors	-.55	-.58	-.56	-.51	-.69	-.56
minownedfile	-.49	-.55	-.55	-.48	-.67	-.49
weakowneds	.44	.47	.49	.35	.61	.50
manager3dir	-.02	.13	.33	.17	.35	.27
manager4dir	.20	.15	.34	.14	.38	.20
manager5dir	.22	.19	.33	.24	.40	.31

Overall, we can observe a strong sense of ownership among all products and for both granularity levels. Surprisingly, the majority of files and the majority of the directories have only one main contributor that does almost all (77-100%) of the commits.

B. Correlations

To show basic relationship between code ownership and code quality, we computed spearman rank correlations between ownership metrics and our code quality measures both described in Section III.D. TABLE III. contains the metric correlation values on source file level. For all products, the metric *ownership* is negatively correlated with the number of bugs. Thus, the more shared the file ownership the higher the likelihood that it will contain code defects. This trend is also supported by the fact that for all projects, the number of *contributors* and the number of *minor* and *minimal* contributors is positively correlated with the number of bugs—the more people contribute to a file's content, the higher the risk of bugs. Organizational ownership metrics on the other hand seem to behave differently for different projects, but are also only weakly correlated with the number of fixed bugs. Nevertheless, it seems that the individual trends of relationships between ownership measurements and the number of bugs are similar for all products, even for the ranking of correlation values. The number of *minor* or *minimal* contributors is for almost all products the measurement with the highest correlation value, followed by *contributors* and *ownership*. As discussed in Section IV, we defined two sets of files: one that contains all files, and one that contains only files that have at least been edited twice. We could not observe a strong difference in terms of correlation values for the two metric sets, as shown in TABLE III.

Correlation values between directory based ownership measures and the corresponding number of bugs fixed per directory are shown in TABLE IV. Similar to correlation values on file level, individual ownership metrics correlate stronger than organizational metrics. Moreover, the percentage of *minor*, *minimal* or *major* contributors among all contributors, and percentage of edits of the minimal contributor, have the strongest correlations (*pcminors, pcminimal, pcmajors, minownerdir*). Whereby, the number of bugs in a directory increases when the percentage of *minor* or *minimal* contributors increases, and the

TABLE V. DETAILS ON PRECISION, RECALL AND F-MEASURE FOR PREDICTING DEFECTIVE SOURCE FILES AND DIRECTORIES. VALUES IN BRACKETS BELONG TO A BASELINE MODEL ONLY BASED ON THE NUMBER OF EDITS PER CODE ENTITY.

	Precision	Recall	F-Measure
Directory level			
OF16	0.77 (0.19)	0.56 (0.04)	0.65 (0.07)
OF15	0.75 (0.19)	0.53 (0.03)	0.62 (0.04)
Ex13	0.75 (0.19)	0.61 (0.27)	0.67 (0.22)
Ex15	0.78 (0.19)	0.60 (0.18)	0.68 (0.19)
O365	0.85 (0.19)	0.71 (0.16)	0.77 (0.18)
Win	0.75 (0.19)	0.59 (0.07)	0.66 (0.10)
File level for files with edits ≥1			
OF16	0.75	0.21	0.33
OF15	0.69	0.21	0.32
Ex13	0.71	0.31	0.43
Ex15	0.74	0.35	0.47
O365	0.79	0.53	0.63
Win	0.70	0.26	0.38
File level for files with edits ≥2			
OF16	0.76 (0.19)	0.31 (0.03)	0.45 (0.05)
OF15	0.70 (0.19)	0.37 (0.06)	0.49 (0.09)
Ex13	0.73 (0.19)	0.38 (0.12)	0.50 (0.15)
Ex15	0.75 (0.19)	0.47 (0.23)	0.58 (0.21)
O365	0.76 (0.19)	0.58 (0.16)	0.66 (0.18)
Win	0.71 (0.19)	0.30 (0.03)	0.42 (0.05)

TABLE VI. METRIC IMPORTANCE FOR PREDICTION MODELS BASED ON OWNERSHIP METRICS CLASSIFYING DEFECTIVE SOURCE FILES.

	OF16	OF15	Ex13	Ex15	OF365	Win
ownership	0.68	0.69	0.73	0.67	**0.85**	0.74
minors	0.35	0.33	0.73	0.34	0.82	0.76
minimals	0.41	0.41	0.35	0.42	0.30	0.33
contributors	**0.69**	**0.70**	**0.75**	**0.69**	**0.85**	**0.77**
manager3	0.41	0.54	0.34	0.42	0.76	0.71
manager4	0.42	0.61	0.68	0.44	0.76	0.65
manager5	0.38	0.62	0.69	0.63	0.78	0.71

number of bugs decreases if more *major* contributors are among the contributors of a directory. Those metrics are followed by the *minowedfile*, *ownershipdir*, *avgownership* and *weakowned* metrics. The lower the ownership for the file with the lowest ownership value (*minonwedfile)* in a directory, the higher the number of bugs. Interestingly, the percentage of edits of the one strongest owner per directory (*ownershipdir*) correlates slightly better with the number of bugs, than the average number for ownership per file (*avgownership*). Also, the higher the percentage of weakly owned files in a directory (*weakowneds*) the higher the number of bugs.

C. Bug Prediction

Some of the ownership correlations discussed in the last section indicate a connection between code ownership on file and directory level and the number of defects fixed in these code entities. However, correlation values have only limited informative value of whether our ownership metrics can be used to build accurate and actionable classification models to identify defective code entities solely based on ownership information. Note, the goal of this paper is not so much to build the best possible defect prediction model, but rather to assess the suitability of ownership metrics to identify defective code entities on different, actionable levels of granularity. The classification model presented in this paper classifies source files and source directories that contained at least one code defect using ownership metrics only.

Classification Performance

TABLE V. shows details on precision, recall, and f-measure for classifying defective source files and directories. As described in Section III.D, we report average values of 10-cross fold experiments. When classifying code directories, metrics *avgminimals* and *avgownership* have been removed as they inter-correlated strongly with other metrics and did not provide any new information value for the classification model.

For the file level classification, we achieve a precision between 0.69 and 0.79 for all files, including single edit files. Classification models ignoring these single edit files show similar precision (between 0.70 and 0.76) However, ignoring single edit files, slightly increases recall values: from [0.21,0.53] to [0.30,0.58]. For the directory level, we see precision values between 0.75 and 0.85 and recall values between 0.53 and 0.79.

In general, we think those prediction results are showing that ownership can play an important factor in terms of code quality. The low recall values on the other hand clearly show that, as suspected, a lack of ownership cannot explain all defects in the systems. However, to use recommendation systems in real world scenarios, a high precision should be preferred over recall. Reporting a high number of false positives would lead to engineers not trusting the recommendation tool and thus ignoring the results in the first place.

To ensure that these classification performance measures stem from ownership metrics rather than churn metrics, we built a baseline model that is solely based on the number of edit per code entity and trained using a random forest (same as ownership models). The performance values for the baseline model are shown in brackets in TABLE V. The baseline model performs significantly worse than the ownership models. Thus, we can conclude that the model performances reported in this paper stem from ownership metrics and not from the correlation between ownership and churn metrics.

Metrics Importance

The metric importance scores on file level are given in TABLE IV. The metric importance scores among all products shows that the number of *contributors* per file is the strongest indicator for defective files. The second most important metric is the percentage of changes that were applied by the main contributor, i.e., *ownership*. The only exception is Windows where the metric is on rank 3, after the number of *minor* contributors. Interestingly, minors is the least important to predict defective files for OF15, OF16, and Ex15. The number of *minimal* contributors is one of the least predictive, ranking last or one but last for all projects except for OF16. On file level, we see that organizational metrics seem more important than some individual ownership metrics. Metrics importance measures for files with at least two edits yield similar results. We omit details due to space restrictions.

Looking at the results for metrics importance on directory level in TABLE VII. , we see that the percentage of edits of the lowest contributor of the file is the most important metrics

TABLE VII. METRIC IMPORTANCE FOR PREDICTION MODELS BASED ON OWNERSHIP METRICS CLASSIFYING DEFECTIVE SOURCE DIRECTORIES.

metrics	Win	OF16	OF15	Ex13	Ex15	O365
avgcontributor	.57	.61	.39	.40	.41	.67
avgownership	.80	.77	.75	.79	.73	.87
ownershipdir	.77	.79	.77	.79	.74	.86
minownerdir	**.84**	**.81**	**.78**	**.82**	**.78**	**.88**
avgminors	.79	.73	.73	.77	.70	.86
avgminimals	.30	.37	.37	.72	.36	.77
pcminors	.82	.79	.77	.80	.74	.87
pcminimal	.78	.27	.30	.78	.28	.83
pcmajors	.82	.79	.77	.80	.74	.87
minownedfile	.81	.78	.75	.79	.73	.87
weakowned	.73	.35	.37	.70	.38	.75
manager3dir	.68	.42	.52	.67	.40	.73
manager4dir	.63	.60	.63	.69	.58	.74
manager5dir	.71	.62	.65	.69	.63	.75

among all products (*minownerdir*). This metric is followed by the percentage of minor (*pcminors*) and major (*pcmajors*) contributors in a directory. After that, there is a variance among the metrics performance and the products. Nevertheless, we see that organizational metrics once again perform among the poorest.

D. Interviews: Weak Ownership

Files that miss a strong owner, i.e., weakly owned files, are not equally distributed throughout the code based – they cluster in few places in the source code base. In fact, 90% of the directories have no weakly owned files. On the other hand, a few directories comprise a large portion of weakly owned files. Weakly owed files have on average 6 times more bugs assigned as files that have a strong owner.

To understand why certain files are weakly owned, and why they cluster in certain directories we used interviews. When we interviewed the engineers, often it was obvious and expected to them that certain files, or files in a certain directory are weakly owned. Those files and directories often can be described to follow a "collective" ownership model. On the other hand, engineers also quickly identified files per name that surprised them to be changed by several engineers.

In Office, strong ownership of code is discouraged. Engineers are working on many different files, whereby they ensure that an area expert is always informed and approves the changes, e.g., by using code review techniques.

There are several reasons why weak ownership occurs. Some of those are because ownership is currently transferred from one person to another or from one team to another. Another can be ongoing refactoring which is performed by another team than the original team that contributed the code, or because of bug fixing. Also, crosscutting concerns or architectural smells can be a reason why several teams have to edit and work on artifacts together.

Not always is the weak ownership expected. With Office, we could see that there are two types of weak ownerships: intentional weak ownership, which was due to the aforementioned reasons, and unintentional ownership. Engineers were not worried about the intentional weak ownership – which might be a form of collaborative ownership, but they were concerned about the unintentional weak ownership for files and directories that they were not expecting that different sets of engineers work on them at the same time. We assume that this might be a form of non-ownership

Similar to the distinction engineers made between files and directories they knew to be weakly owned and the ones they have not been aware of, they also reacted differently to the idea of changing the ownership model. For the files and folders that seem intentionally weakly owned, engineers most of the time did not see the need to change the ownership model. Exceptions were when the weak ownership was due to architectural smells. Here engineers mostly agreed that changes should be made in order to limit the need of several engineers to change the same artifacts. On the other hand, for the artifacts that surprised them, engineers wanted to gather more information why this is happening and then based on that knowledge potentially intervene and act.

VI. DISCUSSION

A. Reflection on Results and Previous Studies.

In this study, we showed that the majority or source code files and directories are strongly owned, supporting the findings of LaToza [11] who showed that at Microsoft there is a strong sense of personal code ownership (72%) and an even stronger sense of team code ownership (92%).

Bird et al. showed that defects correlate with ownership for Windows binaries. In our study, we wanted to know whether such correlations hold for other systems, apart from Windows (see *RQ1* in Section III.A). This is especially important as Foucault et al. [2] showed that the results could not be reproduced for all of the open source systems they investigated. In our study, we investigated four different software systems, i.e., Office, Exchange, Office365 and Windows, and conclude that we could observe that ownership metrics correlated with defects, and that we even could use such metrics to build performant bug prediction models.

In this replication study, we also changed the granularity level of a software component from a binary level to source file and directory level. In contrast to the results of Foucault et al. [2] we could see correlations between ownership metrics on source file and directory level (see *RQ2*). Similar to their findings, did the more coarse-grain directory level metrics perform better than the metrics defined on file level. On the other hand, as our data revealed most of the directories only contain a very small number of files (an average median of 3 files). It is important to remember that many studies showed that defects tend to correlate with size. Therefore, we were particular careful to normalize our metrics with respect to the size of a directory.

Our observed correlation values on file level are on average 0.39 and 0.57 on directory level. The authors in [2] hypothesize that only a correlation value above 0.5 indicates a strong correlation. The interpretation of correlation values is much more complex than that and depends also on the sample size and the significance of the observation. Our sample size was very large on file as well as on directory level, and all spearman correlation showed a very low p value (below 0.01), indicating that the correlations we see are very unlikely to occur due to

chance. Judging the effect size based on Cohen's terms [12], we saw medium to strong effects for the correlation observed even on file level. From that perspective, we conclude that we could observe significant correlations between defects and ownership metrics for all products, and for both granularity levels. In addition, we could observe rather good performances in terms of predicting defective source code files and directories (*RQ3*).

During the interviews and interaction with the product teams, we identified several reasons for a lack of ownership, namely transfer of ownership, bug fixing, refactoring, and architectural issues or smells (*RQ4*). In addition, we observed three groups of ownership: first, the individual ownership, where one engineer is mainly performing changes to and responsible for a code artifact; second collective ownership, where a group of people all together are changing and are responsible and for code artifacts; and finally non-ownership where a lack of accountability and responsibility is observable.

Even though beyond the scope of this paper, we think organizational metrics for example reflecting the number of managers involved can help to separate files that are not owned from files that are collectively owned. We saw that the number of level 3 managers for collective owned files is significantly higher than for files that are not owned. We think further investigation of this phenomena and refinement of the organizational metrics is needed.

B. Recommendations.

In this section, we summarize the lessons learned during this study by formulating several recommendations targeted at development teams.

1. *Do not enforce a strong ownership model without understanding the impact.*

In this study, we showed that weak ownership has an impact on code quality. On the other hand, from the interviews we understand that weak ownership is not always unintended. Simply enforcing a very strong ownership model might not be the right solution, as ownership not only has implications on responsibility and accountability, but also on knowledge and dependencies. For example, if several engineers edit one file, it might be problematic because accountability can be unclear. On the other hand, strong ownership prohibits knowledge transfer. Not being familiar with a certain piece of code is a serious problems and building a mental model about software a tedious task [11]. Also in [13] Mockus showed that it is hard for developers to understand the code of others, and that "developers go to great lengths to create and maintain rich mental models of code that are rarely permanently recorded." This directly impacts the ability of a developer to contribute to source code that she has not contributed before.

2. *Review weakly owned files and directories to understand the mechanisms and dynamics at play (i.e., collaborative ownership or non-ownership).*

An interesting differentiation of type of ownership is done by Martin Nordberg [14]. He makes clear that there is a difference between "collaborative" ownership and "non-ownership". He defines collaborative ownership, as an ownership where code is collectively owned, but responsibilities and schedules are clear. Each team member can work across subsystems as needed. If implemented right, this style helps to build and maintain knowledge about the code among team members and one might expect the quality of such systems to be high. On the other hand, he describes non-ownership as a mode in which several developers make changes to the same system but with minimal accountability for quality or team communication. In such systems, one might expect the quality to be low. During the interviews, engineers explained a similar kind of differentiation between one group of weakly owned files and directories and the other. Engineers seem more worried about the unintended or unknown weakly owned artifacts which seem to correspond to the non-ownership category, and expressed the need to further investigate.

3. *As much as possible assign an owner to currently weakly owned files and directories with unclear accountability.*

We recommend that teams pay attention to ownership of files and directories, and especially to those artifacts that are not intentionally weakly owed, and where the reason for a lack of ownership is unclear. Assigning an owner might not imply that this is the only person that is allowed to change the artifact, but that this person is aware of changes to the artifact for example via code reviewing practices. If the reason for the weak ownership are architectural smell (i.e., cross cutting concerns, god classes etc.) the team should consider refactoring to split weakly owned files into more coherent units.

4. *If driving changes to the ownership model is not possible or desired, use ownership information as indicator of risk.*

For artifacts, where strong ownership is practically not possible to ensure or not desired we highly recommend that changes to such weakly owned files and directories are carefully reviewed. Also, we recommend to use ownership information to drive test efforts.

C. Importance of Replication Studies.

Especially in software engineering, replication studies are limited available and published. Yet, they are crucial to ensure progress in scientific community. Generalizability is often one of the main threats to validity of research studies. As Foucault et al. [2] show in their replication study, the ownership measures do not correlate with all systems under study. On the other hand, we could clearly show that the results generalize beyond the Windows as a subject system.

D. Threats to Validity

Like other empirical studies, ours too has threats to validity.

Not all files touched in bug fixing commit must be necessarily related to the actual fix. Engineers may entangle multiple atomic changes into bigger blobs mixing code changes of different nature, such as fixed and refactorings. Tangled changes may lead to data noise and bias [15, 16].

Using CodeMine [8] as data mining tool to create our data sets, we naturally inherit all threats to validity of CodeMine, this implies likely noise with respect to mapping bug reports to code changes and associations between developers and organizational structure. On the other hand, CodeMine is widely used by practitioners and researchers within Microsoft and data quality is constantly monitored and improved.

In this study, we investigated products using development processes that may contain Microsoft specific elements. The set of products includes products using different processes and having different product objectives. However, all investigated products are Microsoft products. We do not claim our results to be general, supported by the contradicting results reported in [2].

Instead of using one product and relying on one evaluation method (i.e., statistical or interviews) we used triangulation and used four different products and two complementary evaluation techniques to reduces the potential systematic bias that can occur with using only one data source, method, or procedure [17].

VII. Related Work

A. Ownership and Organizational Structure

This paper builds upon work of Bird et al. [1] who discuss the impact ownership on software defects of major software products, i.e., Windows Vista and Windows 7. In their study, especially the effect of changes coming from low expertise developers is examined, and reasons for their contributions are revealed. The authors show that the removal of low-expertise contributions can increase the quality of the software system. Foucault et al. [2] replicated the original study on open source software systems (FLOSS), and found that the findings do not generalize to their open source software under study. In particular, the authors could only find strong correlation between ownership metrics and faults for half of the systems. The authors further investigated this phenomena and come to the conclusion that this is due to the nature of how contributions are distributed in FLOSS projects and also due to the presents of very strong contributors (so called "heroes").

In [18] Posnett et al. reflect on the composition of software systems, and question whether and how the granularity level of software components chosen for investigation impact the outcome of the study results. This concern is truly a valid one, and a key motivator for us to replicate previous studies on ownership on different granularity levels.

Rahman and Devanbu [6] examined the effects of ownership on an even more fine-granular level, the level of contributions to code fragments. The study however uses different ownership measures and focuses purely on open source software systems.

Weyuker et al. [19] focused on the effects the number of contributors have on defect prediction models. The authors found only moderate improvements of fault prediction models that included the cumulative number of developers as prediction factor. Whereby we also use the number of contributors as one factor, we consider a much broader set of ownership metrics. Also, our focus is not to provide a defect prediction model per se, but to show that ownership metrics not only correlate but also can be used to classify defective artifacts.

Meneely et al. [7] studied the effects of the number of contributors on security vulnerabilities focusing on the Linux software system. In contrast to Weyuker et al. [19], the authors showed that a higher number of developers significantly increases the risk of the file to yield a security vulnerability. Files changed by more than 9 developers showed a 16 times higher risk to comprise a security defect.

Herzig and Nagappan [5] examined the impact of organizational structure on test reliability and test effectiveness. In contrast to this study, their study focused on owners of test cases instead of production code and developers.

B. Predicting and Classifying Defect Prone Artifacts

The number of related studies on defect prediction models is large. For the sake of brevity, we only refer to the most relevant related studies in this section. The first studies on predicting defects using code metrics emerged in the 1990s. In 1999, Fenton and Neil [20] provided a comprehensive overview of defect prediction models at that time. In recent years, more reviews on defect prediction models are emerging, e.g. [21, 22]. These reviews show the wide variety of aspects and measurements used for defect prediction purposes. Ostrand et al. [23] used code metrics and prior faults to predict the number of faults. Other studies used change-related metrics [24, 25], developer related metrics [26], organizational metrics [4], process metrics [27], change dependency metrics [28, 29], or test metrics [30, 31] to build defect prediction models, on various software systems and levels of granularity. As previously mentioned, our study does not focus on building a prediction model per se, but focuses on showing a relationship between ownership metrics and code defects.

VIII. Conclusions

In this paper, we replicated and enhanced a study of Bird et al. that looked at the effects ownership has on code quality by measuring the presents of defects. In this study, we extended the previous metrics to be usable on a source file and directory level, thus leading to much more actionable insights. We could show that low ownership metrics correlate with the number of bugs that are fixed either on file, or on directory level. Further, we showed that we can build quite reliable prediction models that can classify files and directories in defective and non-defective entities with an average recall of 0.60 and an average precision of 0.76 for directory level.

As future work, we will use action research to investigate the actions implemented by product teams as a result of this study and also to understand the effects caused by changes to the current ownership model. We plan to do that again by using a combination of data-driven analytics, and close collaboration with product teams.

IX. ACKNOWLEDGMENTS

We thank all product teams for their help and feedback. Special thanks goes out to Carlo Rivera, Alex Gorischek and many others from the Office product team for their active and engaged collaboration. This work is based on data extracted from varies development repositories by the CodeMine.

X. References

[1] 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*, 2011.

[2] 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*, New York, 2014.

[3] C. Bird, N. Nagappan, P. Devanbu, H. Gall and B. Murphy, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," in *Proceedings of the 31st International Conference on Software Engineering*, 2009.

[4] N. Nagappan, B. Murphy and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," in *Proceedings of the 30th International Conference on Software Engineering*, 2008.

[5] K. Herzig and N. Nagappan, "The Impact of Test Ownership and Team Structure on the Reliability and Effectiveness of Quality Test Runs," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014.

[6] 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*, 2011.

[7] A. Meneely and L. Williams, "Secure Open Source Collaboration: An Empirical Study of Linus' Law," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[8] J. Czerwonka, N. Nagappa, W. Schulte and B. Murphy, "CODEMINE: Building a Software Development Data Analytics Platform for Microsoft," *IEEE Software,* pp. 64-71, 2013.

[9] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, pp. 5-32.

[10] M. Kuhn, "caret: Classification and Regression Training," 2011.

[11] T. D. LaToza, G. Venolia and R. DeLine, "Maintaining mental models: a study of developer work habits," in *International Conference on Software engineering* , New York, 2006.

[12] J. Cohen, P. Cohen, S. West and L. Aiken, Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences, Routledge, 2002.

[13] A. Mockus, "Succession: Measuring transfer of code and developer productivity," in *Proceedings of the 31st International Conference on Software Engineering*, Vancouver, 2009.

[14] M. E. Nordberg III, "Managing Code Ownership," *IEEE Softw.,* pp. 26-33 , 2003.

[15] D. Kawrykow and M. P. Robillard, "Non-essential Changes in Version Histories," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011.

[16] K. Herzig and A. Zeller, "The Impact of Tangled Code Changes," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013.

[17] J. Maxwell, "The value of a realist understanding of causality for qualitative research," in *Qualitative inquiry and the politics of evidence*, Walnut Creek, Left Coast Press, 2008, pp. 163-181.

[18] D. Posnett, V. Filkov and P. Devanbu, "Ecological inference in empirical software engineering," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, Washington, DC, USA, 2011.

[19] E. Weyuker, T. Ostrand and R. Bell, "Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models," *Empirical Software Engineering,* vol. 13, no. 5, pp. 539-559, 2008.

[20] N. Fenton and M. Neil, "A critique of software defect prediction models," *Software Engineering, IEEE Transactions on,* vol. 25, pp. 675-689, Sep 1999.

[21] C. Catal and B. Diri, "Review: A Systematic Review of Software Fault Prediction Studies," *Expert Syst. Appl.,* vol. 36, pp. 7346--7354, may 2009.

[22] D. Radjenoviç, M. Heričko, R. Torkar and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology,* vol. 55, pp. 1397--1418, 2013.

[23] T. J. Ostrand, E. J. Weyuker and R. M. Bell, "Where the bugs are," in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, 2004.

[24] R. Moser, W. Pedrycz and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*, 2008.

[25] E. Shihab, A. Hassan, B. Adams and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, Cary, North Carolina, 2012.

[26] M. Pinzger, N. Nagappan and B. Murphy, "Can developer-module networks predict failures?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008.

[27] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, 2009.

[28] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th international conference on Software engineering*, 2008.

[29] K. Herzig, S. Just, A. Rau and A. Zeller, "Predicting Defects Using Change Genealogies," in *Proceedings of the 2013 IEEE 24nd International Symposium on Software Reliability Engineering*, 2013.

[30] N. Nagappan, L. Williams, M. Vouk and J. Osborne, "Early Estimation of Software Quality Using In-process Testing Metrics: A Controlled Case Study," *SIGSOFT Softw. Eng. Notes,* vol. 30, pp. 1--7, may 2005.

[31] K. Herzig, "Using Pre-Release Test Failures to Build Early Post-Release Defect Prediction Models," in *Proceedings of the 25th International Symposium on Software Reliability Engineering*, 2014.

[32] I. H. Witten and E. Frank, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2005.