Distributed Development Considered Harmful?

Ekrem Kocaguneli
Lane Department of CS&EE,
West Virginia University
Morgantown,
WV, USA
ekrem@kocaguneli.com

Thomas Zimmermann,
Christian Bird
Nachiappan Nagappan
Microsoft Research, Redmond, USA
tzimmer@microsoft.com
cbird@microsoft.com, nachin@microsoft.com

Tim Menzies
Lane Department of CS&EE,
West Virginia University
Morgantown,
WV, USA
tim@menzies.us

***Abstract*—We offer a case study illustrating three rules for reporting research to industrial practitioners. Firstly, report "relevant" results; e.g. this paper explores the effects of distributed development on software products.**

Second: "recheck" old results if new results call them into question. Many papers say distributed development can be harmful to software quality. Previous work by Bird et al. allayed that concern but a recent paper by Posnett et al. suggests that the Bird result was biased by the kinds of files it explored. Hence, this paper rechecks that result and finds significant differences in Microsoft products (Office 2010) between software built by distributed or collocated teams. At first glance, this recheck calls into question the widespread practice of distributed development.

Our third rule is to "reflect" on results to avoid confusing practitioners with an arcane mathematical analysis. For example, on reflection, we found that the effect size of the differences seen in the collocated and distributed software was so small that it need not concern industrial practitioners.

Our conclusion is that at least for Microsoft products, distributed development is *not* considered harmful.

I. Introduction

How should the research community talk to industrial practitioners? This papers offers three rules for such communication: *relevance*, *recheck*, and *reflect*.

Regarding *relevance*: From an industrial perspective, successful research comments on important industrial practices. One such important issue in the current software industry is the quality implications of distributed software project (across many sites or many countries). Early results warned that such projects may suffer from lower quality due to geographical dispersion [31] which raises issues of communication [12], and problems building mutual confidence among distributed teams [27]. However, in 2009, Bird et al. checked for those effects in distributed Microsoft projects [4]. They found that management can successful mitigate for these detrimental effects (team members need to be organized along product lines and not on their geographical location).

These relevant results need to be *rechecked*, if subsequent results cast some doubt on prior results. This step may also be required when the context of a result changes (e.g., this study is performed on Microsoft Office instead of Windows). Also the recheck step is optional, when reporting results that have never been reported before. Such rechecking is an important service that researchers can offer industry since industrial practitioners may not have the time to attend all conferences or read all the literature.

In this paper, we rechecked the Bird et al. result regarding the impact of distributed development on software quality. This recheck was necessitated due to the troubling results in 2011 by Posnett et al. [25]. They divided some software defect data into a tree that ranged from the most general to the most specific (from system to sub-system to package to file). They then checked where the best defect models were learned: higher or lower in that tree of data. What they found was that the best defect models learned higher in the tree may not be best for subsets of that data, lower in the tree.

This new results cast some doubt on previous results since Bird et al. had only looked at the big binaries generated by a team but not the smaller source files that built those binaries. Accordingly, this study is a file-level analysis of the effects of collocated vs distributed development. We mined commit and geographical location information of Microsoft Office 2010 developers. We extracted the post-release bug information associated with Office 2010 to comment on the effects of geographical distribution, edit percentages and post-release bug information (at the file level).

At first glance, our results suggested that the 2009 Bird et al. study had indeed fallen into the trap documented by Posnett et al. in 2011. We found that distributed software (studied at the file level) was statistically significantly different than software developed by collocated developers. Note that this is a very alarming result since it calls into question the widespread practice of distributed software development.

Our third rule for communicating research results to industrial practitioners: *reflect* on the results. Research must not confuse industrial practitioners with overly-complicated mathematical results; instead the analysis should help practitioners reflect on the extent of an effect, i.e. whether the observed effect requires an action. For example, while rechecking the Bird et al. results, we compared software built by teams that were either (a) distributed to geographically separate teams or (b) collocated and working together. For this analysis, we employed statistical tests that are standard in the field and which have been accepted, without criticism, in dozens of papers. However, such standard tests used either directly by practitioners or by researchers for practitioner focused analyses *can be very misleading*. When we reflected on our

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Software Engineering in Practice

new results, we found that the real issue was not distributed vs collocated development. Rather the real issue was to correctly communicate effects to industry for the right interpretation:

- It is possible to be misled by the statistically significant difference between collocated and distributed development as indicated by the standard statistical tests.
- However, on reflection, the *size of that effect*, was negligibly small. That is, at least for the software studied here, distributed development is *not* considered harmful.

The rest of this paper is organized as follows. A summary of related work will offer the standard conclusion; i.e. that distributed and collocated development produces different kinds of software of different quality. We will summarize that work as five hypotheses. These hypotheses will be expressed in such a way that (according to the related work, as reported in Section II), we expect them all to be rejected. However, our experiments and analysis of effect size will accept them all: i.e. we can find no major differences between software quality and the kinds of software built by collocated and distributed processes. This will be followed by a discussion on threats to validity. Finally, our future work section poses a question that deserves much attention: Just how many other research "results" are just tiny effects that need not concern industrial practitioners?

II. Related Work

The two aspects explored by this paper are code ownership and distributed development. This section presents our case that these two factors are worthy of exploration.

A. Code Ownership

Human factors play a considerable role in software development [5], [8], [24]. Hence, it is no surprise that there is a considerable literature built around the investigation of code ownership and its relation to software quality.

Rahman and Davenbu use a fine-grained level of analysis on the level of fix-inducing code-fragments to investigate effects of ownership properties on the quality of multiple open-source projects [26]. Their study shows that developer contribution is related to the implicated code in the sense that a developer's specialized experience in a target file is more important than general experience. Bird et al. investigate the ownership properties in two large commercial software products [5] and report that the measures of ownership in both products have a significant relationship with both pre-release and post-release faults. Meneely and Williams investigate Red Hat Enterprise Linux 4 kernel [19], where they empirically investigate the relationship between security vulnerabilities and the developer activities. Their study reports the finding that files with edits from more than 9 developers are 16 times more likely to be vulnerable to security related errors, which is counter intuitive to the Linus' law, which states that given enough beta-tester and co-developer base all the problems can easily be characterized [30].

Boh et al. investigate the data archives of the development work of a telecommunication product for a large time frame of 14 years [6]. They report the interesting finding that specialized experience has the greatest impact on the productivity of individual developers for the modification request completion time. Mockus and Weiss also report a similar finding [21]. They show that changes made by developers, who are more experienced with a software component are less likely to cause failures. Other examples, where the previous development activity of a developer is used as a proxy for ownership and expertise are "Expertise Browser" tool introduced by Mockus et al. [20] and "Expertise Recommender" tool by McDonald et al. [18]. These tools use the number of times a developer has changed a component to measure the amount of ownership and expertise for that particular component.

While the above results are widely cited, they are not universally accepted. There are several counter examples to the relation between software quality and ownership. Weyuker et al. examine the team size information for prediction model performance [33] and report that the addition of ownership related metrics (e.g. cumulative number of developers) only provide a negligible performance improvement. A similar finding is also reported by Graves et al. [11], who show that the number of different developers who have worked on a file does not improve the prediction performance.

B. Distributed Development

For large organizations distributed development is a strategic decision issue related to skill-set availability; the cost of labor; governmental restrictions [4]; various resource constraints [7], [14] and so on. Besides being a good solution for such issues, distributed development turns software development into a more interactive process, where a number of remote development teams have to collaborate in a complementary manner. The collaboration of remote teams brings out a whole set of new problems. There is a wealth of literature defining and tackling the problems associated with distributed development [3], [4], [12], [13], [17], [22], [28]. Some previously studied aspects of distributed development:

- Communication and coordination issues [12], [28];
- Organizational structures of development [22];
- Effects of geographical dispersion [3], [31];
- Possible development strategies (e.g. agile) [29], [32].
- Effects on software quality [9], [27];

Ramasubbu et al. investigate the specific coordination schemes for distributed development [28]. Their claim is that the process frameworks adopted by distributed teams have been developed for collocated contexts, whereas distributed projects require schemes specific for distributed development. Herbsleb et al. investigate the delays of collocated and distributed work items [12] and show that distributed work items take 1.5 to 2 times longer than collocated work items. Bird investigates the organizational structure associated with open source software [5] focusing on the coordination and collaboration schemes between developers. Unlike the prevailing belief, Bird reports that distributed open source collaboration patterns are not haphazard. Spinellis investigates the effects of geographical dispersion in distributed development

of FreeBSD [31]. He reports that geographical dispersion of developers does not have a significant effect on the code quality or on the bug density. Nagappan et al. use a novel approach to dispersion [22]. Instead of geographical, they investigate the relationship of organizational structure to software quality. Their results reveal that organizational metrics are significantly related to post release failure-proneness.

Bird et al. investigate the effects of distributed development on the binaries of Windows Vista [4]. Their findings are aligned with that of Spinellis [31] although the investigated products have different development strategies. Bird et al. report that the geographical dispersion of the commits to binaries has little to no effect on the post release failures.

III. Five Hypotheses

The prior work, as summarized in Section II, helps us identify and check the implications of code ownership and distributed development. Although there are counter examples [11], [33], the code ownership is reported to be strongly associated with software quality [5], [6], [19], [21], [26]. Prior results show that distributed development is affected by the issues of communication and coordination [12], [28] as well as geographical dispersion [3], [31], which may have impacts on the produced software in terms of software quality [4], [27].

A standard way to test the above implications is to form the associated null hypotheses, i.e. to check *their opposite viewpoint*. Forming the null hypothesis and testing its significance is one of the most widely used standard analysis techniques in empirical software engineering [16]. The null hypothesis is basically a statement, which is formed in a way so that it has the possibility of being rejected. Note that the null hypothesis cannot be proven, one can only *"fail to reject"* it. Because, the set of collected data is only a sample, which can help us *reject* a hypothesis, but which is not enough to *prove* it. Once the null hypothesis is formed, it is checked with the appropriate statistical test, which shows us whether we *reject* or *fail to reject* it. If we ignore the minority reports of Weyuker and Graves, then the implications of the related work on code ownership and distributed development can be summarized as follows: *the following five hypotheses will be rejected.*

It is possible that the existence of developers who are experienced on a file is likely to have different impacts on collocated and distributed files. Even if a distributed file has a major developer (who is deemed to be experienced on this particular file), there may still be edits to this file performed by developers located in a remote geographical component. The communication between experienced and inexperienced developers is possibly more difficult in such cases of distributed files. Hence, it may be the case that collocated and distributed files associated with major developers to have different bug counts. This leads to the hypothesis:

H1: Collocated and distributed files associated with major developers (MaDC>0) have similar post-release quality.

The variable $MaDC$ is a measure that detects if there are a small number of developers working on the code. $MaDC$ is the he number of developers, who commit more than 40% of all the edits on a file.

In a similar manner, even if a collocated file has no major developers (i.e. no experienced developer with a considerable percentage of the edits) then if:

- The developers of collocated are still located within close proximity of each other;
- They have better communication and coordination opportunities compared to the developers of a distributed file;

then the files of collocated teams without any major developers (i.e. MaDC=0) may have less bugs compared to distributed files with only minor developers. Hence, the hypothesis:

H2: Collocated and distributed files without any major developers (i.e. MaDC=0) have similar post-release quality.

Prior studies have used binary level information to investigate the relation of distributed development to post-release quality [3], [4]. Their findings show that there is negligible difference between collocated and distributed binaries in terms of post release failures and code metrics. The related hypotheses that will be tested in this research at the file level are:

H3:Collocated and distributed files are equally failure prone.

H4:Collocated and distributed files have similar change and size metrics.

Finally, the ownership properties are shown to be effective information sources for software products [5], [19], [24], [26]. Battin et al. names ownership of a component as one of the critical strategies among the development tasks [2]. In this research we investigate ownership from a distributed development point of view. We need to test if there is a significant difference among different ownership metrics. This leads to our last hypothesis:

H5: Collocated and distributed files have similar ownership metrics.

IV. Methods Used in This Study

This section describes the metrics, data collection, and statistical analyzes used to test **H1, H2, H3, H4, H5**.

A. Definition of Metrics

This section explains the metrics used in our research. So as to position our study accordingly, we used the goal question metric approach as proposed by Basili [1]. We group the metrics used in this research under 4 categories: Distribution, ownership, change and size metrics.

Distribution Metrics: Geographical distribution of a file is defined using 5 divisions: Building, City, State, Country and World. A file can be owned by a group of people working in the same building, same city, same state and so on:

- *Owned By Building (OBB):* If 75% or more of the edits to a file come from a single building, then this variable is 1, indicating that this file is owned at the building level. Otherwise, it is set to 0.
- *Owned By City (OBCi):* When none of the buildings can claim ownership to a file, then we look at the city level. If

75% or more of the edits to a file come from developers located in the same city, then this variable is 1; else, 0.

- *Owned By State (OBS):* If 75% or more of the edits come from the same state, then OBS is 1; otherwise 0.
- *Owned By Country (OBCo):* If 75% of the edits to a file come from the same country, then 1; otherwise, 0.
- *Owned By World (OBW):* If no geographical component can own up to 75% of the edits, then file OBW = 1.

The rationale for the above metrics is as follows. These metrics indicate the smallest geographical entity, in which the developers account for 75% of the edits to a file. Note that distribution metrics are exclusive, a file can be owned at only one geographical division. The geographical division at which a file is owned influences how much a file is affected by the issues of distributed development [4]. For example, if a file is owned at the building level, then the developers are walking distance apart from one another. On the other hand, if it is owned at the country level, then the developers are relatively more affected by communication issues inherent in distributed development. These metrics enable us to define different scenarios of distributed development (see Section IV-C for 4 different scenarios). The reason of choosing a threshold of 75% of the edits for the distribution metrics is twofold. First reason is based on the prior research on organizational [22] as well as geographical distribution [3], [4], where the same threshold value has been utilized. Also, a sensitivity analysis of threshold values from 65% to 85% with increments of 5% yielded no dissimilar results.

Ownership Metrics:

Edit Frequency (EF): The total number of times a file is edited. An edit can be defined as the activity of a developer checking out a file, performing changes on this file and committing it back. Note that since EF forms the basis for multiple other ownership metrics, it is listed under ownership metrics in this study. However, EF can also be used as a basis for organizational metrics, .e.g. Nagappan et al. lists EF as an organizational metric [22].

The purpose of EF is twofold. Firstly if a file is being edited too many times, then it can be an indicator that the file is unstable or that the file is likely to have issues of post-release stability. Secondly, EF serves as a basis for the following ownership and experience metrics.

Major Developer Count (MaDC): The number of developers who commit more than 40% of all the edits on a file.

The rationale for this metrics is that specialized component-based experience is important [26] and developers with more experience on a component are less likely to cause failures [21]. That is, major developers with higher edit percents on a file are deemed to have expertise on that file.

Top Ownership Percentage (TOP): The percentage of edits by the developer, who has done most of the edits.

Similar to the major developer count, the value of the highest edit percentage on a file can inform us about the expertise of the developers working on a particular file.

Total Developer Count (ToDC): The number of all the developers editing a file.

The total number of developers working on a file is also important for a component [19], [33]. We are interested to see if collocated and distributed files are edited by a considerably different number of developers.

Change Metrics:

- *Total Added LOC (ALOC):* Total LOC added to a file.
- *Total Deleted LOC (DLOC):* Total LOC file deletions.
- *Total Edited LOC (CLOC):* Total LOC edited in a file.

We use these change metrics since the total added, deleted and edited lines of code in all edits to a file can convey information that is useful to observe files that experience bigger amounts of change, hence become more susceptible to post-release problems.

Size Metrics:

- *Number of Functions (NOF):* Functions per file.
- *Number of Classes (NOC):* Classes per file.

We use these size metrics since the total number of functions and classes in a file inform us about the functional properties of a file. A file with a much higher number of functions and classes may be more likely to have post-release issues. Hence, we include these metrics into our analysis.

B. Data Collection

The data collected for this study enables us to investigate post release quality of a distributed project at file level. The collected data also enables us to test hypotheses regarding the code ownership properties in collocated and distributed files as well as their relation to post release quality. This study examines *Office 2010*. Office 2010 is developed within Microsoft by a total of 1500+ developers. It is composed of tens of thousands of source code files. The development history is traced from the release to manufacture (RTM) date of Office 2007 until the RTM date of Office 2010.

For our research there was the need for various different types of data, among which the most important was the commit information. The software commit information of Office 2010 (as well as other products) is stored in the software repositories of Microsoft. The repository data contains information regarding the name of the developer who performed the change, the time-stamp of the change, which file and what lines of this file were changed, what the intention of the change was (e.g. development for a new feature, enhancement, fixing a bug etc.), on which branch this change was performed and so on. Note that there is a complex branch structure associated with Office 2010. Some of the changes performed on files are associated with the movement of development activity towards the main branch (i.e. trunk). Such activities are not related to development, hence they are ignored.

For our analysis, we collected the number of edits performed by each developer on Office 2010 source files. We also recorded the change metrics associated with each edit. Then we mined the geographical information of each developer's location to map files to different geographical components.

To collect the post-release bug information, we traced the bug correction activity of Office 2010 from RTM until the

release date of service pack #1 (SP1). Each bug correction activity is associated with a collection of file changes (so called *"change-list"*). The file and bug information is mined from the change-lists associated with post-release bugs. Bug and the development activity information yields the final form of our data, where for each file we have the distribution, ownership, change and size metrics as well as the bug count information.

C. Scenarios and Discovering the Effects

For the first part of the analysis, we are interested in discovering the effects between collocated and distributed files, i.e. is there a statistically significant difference? Collocated and distributed file definition can vary according to different scenarios that we are interested in. With the 5 geographical components (building, city, state, country and world), we have 4 scenarios of collocated and distributed files:

Scenario	Collocated	Distributed
BLD	OBB files	All except collocated files
CTY	OBB and OBCi files	All except collocated files
STT	OBB, OBCi and OBS files	All except collocated files
CNT	OBB, OBCi, OBS and OBCo files	Only the OBW files

The scenario names are the first three consonants of the biggest geographical location of the collocated files. In the first scenario the biggest geographical component of the collocated files is *building* and the scenario is referred as *"BLD"*. Table I shows the distribution of all files into collocated and distributed file groups according to each different scenario.

TABLE I
THE PERCENTAGE DISTRIBUTION OF FILES IN EACH SCENARIO.

Scenario	% Collocated	% of Distributed
BLD	69.8%	30.2%
CTY	89.2%	10.8%
STT	93.6%	6.4%
CNT	96.9%	3.1%

D. Statistical Analysis

For the following analysis, we compared populations using the non-parametric Wilcoxon rank-sum test (a.k.a. Mann-Whitney U test) with 95% confidence for hypothesis testing. The results are presented in terms of the standard p measure; i.e. what is the probability that the conclusion is wrong? The standard rule is to use $p < 0.05$ since this implies less than a 5% probability of a mistake in that conclusion.

Kampenes et al. note that the conclusions drawn merely from statistical tests and the p values may be erroneous, if statistical test is not complemented with an *effect size* analysis [15]. They argue that the mere use of p-values is insufficient for decision making. Considering the effect sizes is beneficial not only to report meaningful outcomes of the experiments, but also for comparison purposes [10], [15].

After an extensive literature review, Kampenes et al. endorse the Hedge's g effect size test. The formula of Hedges' g is

$$g = \frac{\bar{X}_1 - \bar{X}_2}{s_p} \quad (1)$$

TABLE II
THE SIZE CATEGORIES FOR THE EFFECT SIZE OF SE STUDIES AND CORRESPONDING EFFECT SIZE INTERVALS FOR HEDGES' g. FROM [15].

Size Category	g
Small	0.17
Medium	0.60
Large	1.40

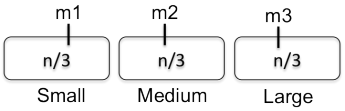

Fig. 1. Given n cases, we have n-many g-values, which are sorted and divided into 3 equal size bins. Medians of each bin (m1, m2, m3) indicate the mid-points (as given in Table II) of small, medium and large effects.

where $\bar{X}_1$ and $\bar{X}_2$ are metric value sample means of collocated and distributed files. s_p is the pooled standard deviation based on the standard deviations of two populations:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 - 1) + (n_2 - 1)}} \quad (2)$$

where n_1, n_2 are sample sizes and s_1, s_2 are standard deviations of the first and the second population, respectively.

The g-value can be negative or positive depending on which population is assigned to X_1 and which one is assigned to X_2. For uniformity, we always subtracted collocated file population metrics' mean from that of the distributed file population metrics. This way, for each metric we can see greater or smaller than relationship (if positive, then metric's value for distributed development is larger) between collocated and distributed files as well as the extent of the relationship (small, medium or large).

Kampenes et al. review 92 manuscripts to derive standardized effect sizes and divide the sorted effect size values into equal size bins, which are used to derive the midpoints of small, medium and large effect sizes. The g-value standard conventions as given by Kampenes et al. for SE are presented in Table II. Assuming that we have n cases (in Kampenes et al.'s case $n = 92$) to calculate the effect sizes, then we would have n-many g-values. These values are ordered in ascending order and divided into 3 equal size bins, see Figure 1. The median values m1, m2 and m3 of Figure 1 indicate the mid points of small, medium and large effect sizes, respectively.

Kampenes et al. note the importance of contextual effect sizes and that a *"ritualized interpretation"* should be avoided [15]. In the ideal case, when there are enough contextual samples, one may choose to calculate the context specific g-values. Since this study uses a single product, we will make use of the effect size values from the close context of SE, derived from 92 SE studies [15].

V. RESULTS

Recall from the above that related work suggests we should reject the five hypotheses defined in this study. In summary, that did not happen:

- **H1** and **H2** were both failed to be rejected.
- Initially, our statistical hypothesis tests rejected **H3,H4,H5** but the Hedges test showed that the size of these effects were negligible. Therefore we had to accept these three hypothesis.

That is, for the software explored here, none of the following results call into question the value of distributed development.

A. Hypothesis Testing Results

For completeness sake, we first present the hypothesis testing results. Then, we show that the somewhat arcane mathematics of hypothesis testing actually confuse a very simple result for practitioners: i.e. the size of the effect between collocated and distributed software is negligibly small.

Hypothesis testing failed to reject **H1: Collocated and distributed files associated with major developers (MaDC>0) have similar post-release quality.** Before the comparison of files to test **H1**, we identified the collocated and distributed files for which there is a major developer (i.e. $MaDC > 0$). Then, in a similar fashion, we identified the ones without a major developer (i.e. MaDC=0). Testing *H1* requires the comparison of the post release bug counts associated with the collocated and distributed files for which $MaDC > 0$. The *p*-values of this comparison for all 4 scenarios are given in Table III. Note that all the *p*-values are greater than 0.05, i.e. collocated and distributed files with a major developer do not have statistically significantly different bug counts.

Also, hypothesis testing failed to reject **H2: Collocated and distributed files without any major developers (i.e. MaDC=0) have similar post-release quality.** To see this, note that testing **H2** requires the comparison of the bug counts of files without a major developer, i.e. the collocated and distributed files for which $MaDC = 0$. The *p*-values of this comparison for 4 different scenarios are provided in Table IV. In all the comparisons the *p*-values are much bigger than 0.05.

Hypothesis testing rejected **H3: Collocated & distributed files are equally failure prone.** To see this, for each of the 4 scenarios (defined in Section IV-C), we compared the bug numbers of collocated and distributed files. The corresponding *p*-values of the comparisons are given in Table V. Note that in all of the scenarios the bug counts of collocated and distributed files are significantly different.

The files without a bug (i.e. files with a bug count of zero) are a big majority for Office 2010. Since such a distribution might influence the result of the statistical test, we performed the same statistical test on collocated and distributed files (whose bug counts are greater than zero). The results were the same: all *p*-values very close to zero. Our analysis showed that for multiple scenarios, distributed and collocated files have significantly different post release bug counts.

Similarly, hypothesis testing rejected **H4: Distributed and collocated files have similar change and size metrics.** To see this, we compared the change and size metrics of distributed and collocated files for 4 different scenarios. The resulting *p*-values for each scenario can be seen in Table VI. The *p*-values of Table VI show that collocated and distributed files are different to one another in terms of every size and change metric, with the exception of the 4 bold-face cells. For most of the our comparisons the collocated and distributed files do not have similar change and size metrics.

Lastly, hypothesis testing rejected **H5: Distributed and collocated files have similar ownership characteristics.** As evidence for this, consider the *p*-values associated with the comparison of collocated and distributed files in terms of ownership metrics (in Table VII). For 4 different scenarios and 4 metrics, there is no *p*-value that is greater than 0.05, i.e. for all the comparisons, the collocated and distributed files have significantly different metric values.

TABLE III
ABOUT **H1**: THE *p*-VALUES OF POST RELEASE BUG COUNT COMPARISON BETWEEN COLLOCATED AND DISTRIBUTED FILES, FOR WHICH $MaDC > 0$.

Scenario	*p*-value
BLD	0.24
CTY	0.74
STT	0.47
CNT	0.41

TABLE IV
ABOUT **H2**: THE *p*-VALUES OF POST RELEASE BUG COUNT COMPARISON BETWEEN COLLOCATED AND DISTRIBUTED FILES, FOR WHICH $MaDC = 0$.

Scenario	*p*-value
BLD	0.65
CTY	0.83
STT	0.64
CNT	0.38

TABLE V
ABOUT **H3**: THE COMPARISON OF COLLOCATED AND DISTRIBUTED FILE BUG COUNTS.

Scenario	*p*-value
BLD	0.00
CTY	0.00
STT	0.01
CNT	0.03

TABLE VI
ABOUT **H4**: THE *p*-VALUES OF COMPARING THE CHANGE AND SIZE METRICS OF COLLOCATED AND DISTRIBUTED FILES.

Scenario	ALOC	DLOC	ELOC	NOF	NOC
BLD	0.00	0.02	**0.40**	**0.89**	**0.19**
CTY	0.00	0.00	**0.56**	0.00	0.00
STT	0.00	0.00	0.00	0.00	0.00
CNT	0.00	0.00	0.00	0.00	0.00

TABLE VII
ABOUT **H5**: THE *p*-VALUES OF COMPARING THE OWNERSHIP METRICS OF COLLOCATED AND DISTRIBUTED FILES.

Scenario	EF	TOP	ToDC	MaDC
BLD	0.00	0.00	0.00	0.00
CTY	0.00	0.04	0.00	0.01
STT	0.00	0.00	0.00	0.00
CNT	0.00	0.00	0.00	0.00

TABLE VIII
THE EFFECT SIZES OF THE DIFFERENCE BETWEEN COLLOCATED AND DISTRIBUTED FILES IN TERMS OF HEDGES' g AND THE CORRESPONDING SIZE CATEGORY [15]. g IS CALCULATED FROM EQUATION 1 AND THE EFFECT SIZE IS CALCULATED FROM TABLE II.

	BLD		CTY		STT		CNT	
Metric	g	effect size	g	effect size	g	effect size	g	effect size
Bug Count	0.05	small	0.05	small	-0.01	small	0.00	small
ALOC	-0.01	small	-0.01	small	0.01	small	0.06	small
DLOC	-0.01	small	-0.01	small	0.01	small	0.05	small
ELOC	-0.01	small	0.00	small	0.01	small	0.04	small
EF	0.07	small	0.11	small	0.19	small	0.02	small
ToDC	0.34	small	0.05	small	0.10	small	-0.01	small
TOP	-1.09	LARGE	0.05	small	0.08	small	0.15	small
MaDC>0 Bugs	-0.01	small	-0.02	small	-0.02	small	-0.01	small
MaDC=0 Bugs	0.02	small	0.26	small	0.29	small	0.20	small
NOC	0.02	small	-0.03	small	-0.03	small	0.01	small
NOF	0.09	small	0.03	small	-0.07	small	-0.05	small

A factor that may influence ownership metrics is the number of developers working on files. For some files, it is possible that all the edits to a file come from a single developer. That is not necessarily a problem, after all one developer may be the sole owner of a file; however, all such files would be owned at the building level and never have the possibility of being a distributed file. We repeated the ownership metrics analysis on the files with at least 2 developers (i.e. any file could be collocated or distributed). The results were the same: the p-value tests showed that collocated and distributed files were significantly different from one another in all cases.

B. Effect Size Results

The rejection of **H3,H4,H5** is consistent with much of the prior literature on code ownership, distributed development, and code quality. However, a closer reflection over the data shows that the that rejection was premature. Figure 2 shows the bug counts of collocated and distributed files (in BLD scenario) from 0^{th} to 100^{th} percentile, with increments of 1. Note that the difference between the distributed and collocated distributions, while statistically significant, is very slight.

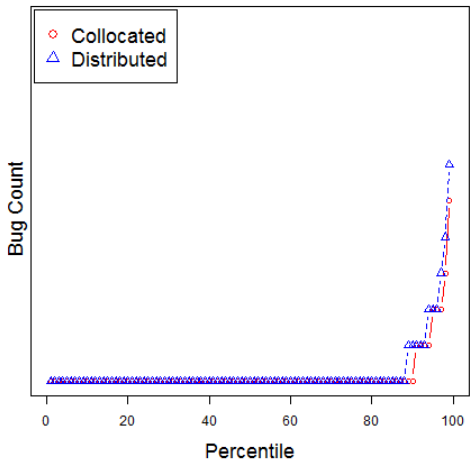

Fig. 2. Bug counts of collocated and distributed files from 0^{th} to 100^{th} percentile with increments of 1.

The Hedges test confirms our visual impression of Figure 2: i.e the differences in the bug counts is very slight. The first line of Table VIII reports the results of the Equation 1 calculation for the bug count. The other lines of that table show a similar calculation for our source code measures. Note that the standardized mean difference effect sizes are mostly categorized as small. The only counter example is the TOP metric comparison for BLD scenario, where the effect size is categorized to be LARGE.

Therefore, even though statistical hypothesis testing rejected three of our hypotheses, we must conclude that these effects are small and negligible.

That is, for this sample of proprietary Microsoft software, issues of code ownership and distributed development do not impact:

- Code quality (measured in number of defects);
- The kinds of code developed (measured in terms of the static code measures shown above).

In summary, we can find no difference that distributed development is harmful in this code since there is practically no difference between software developed by collocated and distributed software teams working on Microsoft Office 2010.

VI. THREATS TO VALIDITY

The metrics used in this research are collected through automated SE tools that are used in production. Hence, we do not see considerable construct validity issues concerning huge errors in measurement.

Internal validity assures that the variations in the dependent variable can be attributed to the independent variables [23]. Wright et al. reports that confounding factors and selection bias are among the fundamental threats to internal validity of SE studies [34]. This is particularly true for analyses performed on large software products. Because, all the confounding factors may not be known in advance and the selection of metrics is biased by the availability of the metrics. Our research is no exception. For example, the selection of the metrics was limited by the availability of these metrics for Office 2010 (e.g. no test coverage or dependency metrics). Also, in this study we were interested in the raw EF values

made to files and since we observed that distributed and collocated files have similar class and function counts (NOF and NOC respectively), we did not normalize the EF values by the LOC of files. Given that file sizes differ considerably, this can be a confounding factor.

One of the most obvious threats to the validity of the results presented in this research is the external validity. External validity means that the results of a study can be generalized for different settings [23]. We have no reason to posit that the results presented here would be similar to a replication study. On the other hand, the proposed methodology can be applicable to different projects regardless of the choice of organization or particular metrics. Although this research investigates a different product and different metrics at a finer granularity level in comparison to previous distributed development studies at Microsoft [3], [4], there are similarities in terms of conclusions. Previous studies report that distributed development has little to no effect on post release quality. Our results indicate that although the differences are statistically significant, the effect sizes are limited.

VII. Discussion

Section V concluded that a supposed effect was so small that it was negligible. A reader of this paper might look at Figure 2 and exclaim "Of course! That was obvious!". Yet a practitioner or a researcher merely using hypothesis testing could have been mislead by the statistics, if the size of the effect was not properly reflected upon. Our concern is that they would not be the only ones being mislead. In fact, we fear that lack of effect size consideration is a widespread error that may threaten the recommendations to industry. In their extensive review of effect sizes in software engineering, Kampenes et al. [15] report that less than 30% of 90+ research results they studied reported effect size results. Questioning the size of the observed effects is critical, because recommendations to practitioners help them make decisions, such decisions should be guided not only by existence of an effect (observed through hypothesis tests) but also by the size of the that effect. Because, an existing effect may be significant yet so small (as in the case of ownership and distributed development metrics of this study) that the practitioners need not be concerned to take an action. Furthermore, the analysis patterns proposed by the academia may be adopted by practitioners on a day to day basis. We show that widely used hypothesis testing should be complemented with effect size analysis and we propose a recommended effect size analysis to practitioner audience.

It is possible that many "effects" reported in the SE literature are actually small and ignorable. For future work, we strongly advise a recheck and reflection on the past results and recommendations to see just how many of those results disappear as small effects. We also strongly advise reporting the effect size in research results as they provide further interpretation possibility for industry whether or not to be concerned about the observed effect. As shown in Section IV-D, this effect size test is quite simple to be applied.

Another issue that requires discussion is that our results are different from the standard view as reported above in Section II. In the standard view, distributed systems are expected to be very different from those developed in collocated manner. But in our results, they are not. Why?

The first comment to be made here is that our results are not without precedent. Recall that Weyuker and Graves [11], [33] reported that code ownership and number of developers did not change the quality of the systems that they studied. Also, working on different data to that studied here, reported that distributed development does not necessarily damage software quality.

The second comment to be made is that many of the other prior studies were based on open source systems. Perhaps software built by distributed teams at Microsoft fares well enough that intra-organizational development can maintain the standards required for quality development. This conjecture is a small variant of the Bird et al. [4] conclusion and we plan to explore this matter further in future work.

VIII. Conclusion

This paper offers three rules on how researchers should communicate results to industry: *relevancy, recheck* and *reflect*. We studied if collocated and distributed teams produce different kinds of software. Given the current globalization of the software industry, this is a topic of great current *relevancy*.

Old relevant results need to be *rechecked* when new papers offer results that challenge prior results. Note that such rechecks are important service that researchers can offer the industrial community. Industrial practitioners are busy people racing to meet their development schedules. Hence, these practitioners may not have the time to read the journal papers or attend the conferences. For example, it is unlikely that many industrial developers have heard of the Posnett et al. results that say conclusions that hold at one scale (e.g. for code binaries) may not hold for another scale (e.g. the smaller code files used to build the bigger binaries). Hence, the recheck of this paper was to check if the optimism of Bird et al. (that management actions can mitigate for some negative effects of distributed development) were still valid at the file level.

Once results are generated, it is important to *reflect* on the new results to observe the size of an effect besides its statistical significance. For example, hypothesis testing reported that there were significant differences between distributed and collocated software, yet the *size of the effect* between collocated and distributed development is small enough to be ignorable. In this research we propose practitioners a recommended method [15] to observe the size of the effects, which can aid them better reflect on the observed effects. As a result of this study, we saw that the distributed development is not considered harmful since, at least for the software studied here, we found no difference in the metrics we collected from collocated and distributed software projects.

IX. Acknowledgments

We would like to thank Microsoft Engineering Excellence team for their support in this research. We would also like to

thank Brendan Murphy, Rich Hanbidge and Jacek Czerwonka for providing us with their valuable feedback and domain knowledge. Ekrem Kocaguneli conducted this research during a summer internship at Microsoft Research Redmond in 2012.

REFERENCES

[1] V. R. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, University of Maryland at College Park, College Park, MD, USA, 1992.

[2] R. Battin, R. Crocker, J. Kreidler, and K. Subramanian. Leveraging resources in global software development. *Software, IEEE*, 18(2):70 –77, mar/apr 2001.

[3] C. Bird and N. Nagappan. Who? where? what? examining distributed development in two large open source projects. In *MSR*, pages 237–246, 2012.

[4] C. Bird, N. Nagappan, P. T. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality? an empirical case study of windows vista. In *ICSE*, pages 518–528, 2009.

[5] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. T. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *SIGSOFT FSE*, pages 4–14, 2011.

[6] W. F. Boh, S. A. Slaughter, and J. A. Espinosa. Learning from experience in software development: A multilevel analysis. *Management Science*, 53(8):1315–1331, August 2007.

[7] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *Software, IEEE*, 18(2):22 –29, mar/apr 2001.

[8] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 353–362, New York, NY, USA, 2006. ACM.

[9] J. Cusick and A. Prasad. A practical management and engineering approach to offshore collaboration. *Software, IEEE*, 23(5):20 –29, sept.-oct. 2006.

[10] T. Dybå, V. B. Kampenes, and D. I. Sjø berg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, Aug. 2006.

[11] T. Graves, A. Karr, J. Marron, and H. Siy. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653 –661, jul 2000.

[12] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481 – 494, june 2003.

[13] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, FOSE '07, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.

[14] H. Holmstrom, E. O. Conchuir, P. J. Agerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 3 –11, oct. 2006.

[15] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. Systematic review: A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.*, 49(11-12):1073–1086, Nov. 2007.

[16] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721 – 734, aug 2002.

[17] R. Kommeren and P. Parviainen. Philips experiences in global distributed software development. *Empirical Software Engineering*, 12:647–660, 2007. 10.1007/s10664-007-9047-3.

[18] D. W. McDonald and M. S. Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 231–240, New York, NY, USA, 2000. ACM.

[19] A. Meneely and L. A. Williams. Secure open source collaboration: an empirical study of linus' law. In *ACM Conference on Computer and Communications Security*, pages 453–462, 2009.

[20] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 503–512, New York, NY, USA, 2002. ACM.

[21] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5:169–180, 2000.

[22] N. Nagappan, B. Murphy, and V. R. Basili. The influence of organizational structure on software quality: an empirical case study. In *ICSE*, pages 521–530, 2008.

[23] D. E. Perry, A. A. Porter, and L. G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 345–355, New York, NY, USA, 2000. ACM.

[24] 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*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM.

[25] D. Posnett, V. Filkov, and P. T. Devanbu. Ecological inference in empirical software engineering. In *ASE*, pages 362–371, 2011.

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

[27] N. Ramasubbu and R. K. Balan. Globally distributed software development project performance: an empirical analysis. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 125–134, New York, NY, USA, 2007. ACM.

[28] N. Ramasubbu and R. K. Balan. Towards governance schemes for distributed software development projects. In *Proceedings of the 1st international workshop on Software development governance*, SDG '08, pages 11–14, New York, NY, USA, 2008. ACM.

[29] B. Ramesh, L. Cao, K. Mohan, and P. Xu. Can distributed software development be agile? *Commun. ACM*, 49(10):41–46, Oct. 2006.

[30] E. Raymond. The cathedral and the bazaar. *Knowledge, Technology and Policy*, 12:23–49, 1999. 10.1007/s12130-999-1026-0.

[31] D. Spinellis. Global software development in the freebsd project. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, GSD '06, pages 73–79, New York, NY, USA, 2006. ACM.

[32] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, page 274a, jan. 2007.

[33] 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*, 13:539–559, 2008. 10.1007/s10664-008-9082-8.

[34] H. K. Wright, M. Kim, and D. E. Perry. Validity concerns in software engineering research. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 411–414, New York, NY, USA, 2010. ACM.