# Does geographical distance effect distributed development teams: How aggregation bias in software artifacts causes contradictory findings

Thanh H. D. Nguyen[1], Bram Adams[2], Ahmed E. Hassan[1]
[1]SAIL, School of Computing, Queen's University
557 Goodwin Hall, Kingston, Ontario, Canada, K7L 3N6
[2]MCIS, Ṕolytechnique Montréal
2500 Chemin de Polytechnique, Montréal, Québec, Canada, H3T 1J4
Email: {thanhnguyen,ahmed}@cs.queensu.ca,bram.adams@polymtl.ca

***Abstract*—Does geographic distance affect distributed software development teams? Researchers have been mining software artifacts to find evidence that geographic distance between software team members introduces delay in communication and deliverables. While some studies found that geographical distance negatively impacts software teams, other studies dispute this finding. It has been speculated that various confounding factors are the reason for the contradicting findings. For example, newer tools and practices that enable team members to communicate and collaborate more effectively, might have negated the effects of distance in some studies.**

**In this study, we examine an alternate theory to explain the contradicting findings: the different aggregations of the software artifacts used in past studies. We call this type of bias: the aggregation bias. We replicated the previous studies on detecting the evidence of delay in communication using the data from a large commercial distributed software project. We use two different levels of artifacts in this study: the class files and the components that are the aggregation of the class files. Our results show that the effect of distance does appear in low level artifacts. However, the effect does not appear in the aggregated artifacts. Since mining software artifacts has became a popular methodology to conduct research in software engineering, the result calls for careful attention in the use of aggregating artifacts in software studies.**

## I. Introduction

Global software engineering brings many benefits to software companies [1], [2], [3]. Being close to the customers while maintaining access to cheap and skilled global labour markets gives large software companies a competitive advantage while maintaining the same level of service to customers.

However, distributed global software engineering poses new problems compared to a traditional collocated setting [1], [3], [4]. Surveys of distributed teams reveal difficulties due to the geographic distance such as delay in communication, coordination breakdowns, and personnel conflicts due to cultural differences. These challenges threaten to reduce or even overrun the benefits of distributed development.

The quest to find empirical evidence of the effect of distance on software artifacts, however, has yielded conflicting findings. Various research teams [5], [6], [7], [8], [9] reported that distributed development has a negative impact on productivity. On the other hand, other teams [10], [11], [12] found that the effect of distance is negligible. One possible explanation is that different companies have different strategies [13] to mitigate the effect of distance.

However, in recent years, many studies have been examining the existence of various types of bias [14], [15], [16], [17], [18], [19], [20], [21], [22] in software artifacts that can cause contradicting findings in empirical studies. We suspect that a particular kind of bias, aggregation bias [23], is the cause of the contradicting findings on the distance effect. Aggregation bias happens when one evaluates the same hypothesis using a particular software artifact (e.g., Java classes) and the aggregation of those artifacts (e.g., Java packages), and the findings contradict. Empirical evidence of aggregation bias is reported in many studies [17], [19], [20], [21].

In this paper, we conduct a case study on a large-scale distributed commercial software project: IBM Rational's Jazz™[1]. We explore if aggregation bias is a potential cause of the contradicting findings on the distance effect by analyzing the evidence of the distance effect on two different artifact levels: components and source files. We find that:

- Distance has an impact on productivity when studied at the file level. Distributed source files take longer to patch and contain more defects even when the effect of the confounding factors are controlled.
- Distance has a negligible impact on productivity when studied at the component level. There is a difference in defect count between distributed and collocated components. However, when we factor in the confounding factors, the difference disappears.

The result shows that aggregation bias is indeed a potential cause for the contradicting findings in literature on the distance effect. Thus, researchers should always explore and test their hypotheses on different levels of software artifacts explicitly.

The structure of this paper is as follows. In the next section, we discuss the conflicting findings of prior research, which explored the effect of distance in distributed development.

[1]http://www.jazz.net. IBM and Jazz are trademarks of IBM Corporation in the US, other countries, or both.

TABLE I
PAST STUDIES ON THE DISTANCE EFFECT

| **Study** | **Artifacts used** | **Target of investigation** | **The distance effect** |
|---|---|---|---|
| [8] | Change request (Task or Defect) | Delay in communication | Distributed ⇒ longer to fix |
| [11] | Change request (Task or Defect) | Delay in communication | Negligible effect |
| [10] | Component (DLL) | Number of defects | Negligible effect |
| [5], [6], [7] | Projects | Number of defects | Distributed ⇒ more defects |
| [12] | Source code (File level) | Number of defects | Negligible effect |
| [9] | Change request (Task or Defect) | Delay in communication, chance of reciprocation | Mixed |

We also introduce the research questions used to structure our analyses. Section III presents our data collection method, the constructs and the statistical techniques used in this study. Section IV and Section V present the results of each research question. We discuss our results in Section VI. We conclude in Section VIII.

## II. Related Work and Motivation

In this section, we introduce related work on the effect of distance in distributed software teams. We also explain the different levels of artifacts often used in previous studies.

### *A. Prior research on the effect of distance in global software engineering projects*

The effect of distance in distributed software teams, which we will call "the distance effect" for short in this paper, has been studied extensively. Experiments in the laboratory have found that even though technologies have enabled workers to collaborate remotely, the effectiveness compared to collocated workspaces is questionable [24]. Empirical studies of distributed software teams have explored difficulties of such workplaces. Battin et al. [1] report that distributed teams at Motorola suffer from loss of communication richness, coordination breakdown, and geographic dispersion. A survey of distributed software teams at Lucent Technologies showed that there are perceived difficulties in communication when it comes to distributed development [8]. Hinds et al. [25] found that distributed teams report more conflicts. Studies with large datasets have identified different factors such as time-zone differences [26], personal imbalance, or differences in experience [27] that affect productivity and quality of distributed teams.

However, the quest to find hard evidence through analyses of software artifacts has yielded contradicting findings. We summarize the past studies on the distance effect in Table I. For each study, we show the reference to the paper, the artifacts used in the study, and the findings. Herbsleb et. al. [8] found that the distance effect increases the delay in communication. However, our past case study [11] on another software project finds that the effect is very small. Bird et. al. [10] found that the number of defects does not change with the increased distance. Cataldo et. al. [5], [6], [7], on the other hand, found the opposite. Wagstrom and Datta [9] study the communication speed and reciprocality in a large distributed team. They found that geographic distance has no effect on both factors but the time zone difference has an impact on communication speed.

There are many possible explanations for the conflicting findings. One possibility is that companies have different strategies [13] to mitigate the effect of distance. For example, Herbsleb et. al. [8] and Nguyen et. al. [11] studied delay in task completion and communication at Lucent and IBM Jazz about five years apart from each other. It is possible that the Jazz project was able to cope with the geographic separation because of new awareness and new strategies for global software engineering.

However, we suspect that the different levels of software artifacts used in these studies might also play a role. We can observe from Table I that each past study focuses on a single level of artifacts. The authors probably looked at the most natural and available level for their studies. Herbsleb et. al. [8] and Nguyen et. al. [11] used change requests. Bird et. al. [10] and Cataldo et. al. [5], [6], [7] used software components or software projects. Spinellis [12] analyzed files.

### *B. Aggregation bias*

Figure 1 shows a conceptual diagram of a typical study that uses software artifacts. On the left side, we have the subject of study, in which the researchers are interested. For example, we studied the use of call dependencies on defect prediction [19]. In the middle are the artifacts that researchers collect. On the right side are the possible conclusions of the analysis. For example, we tried to find the effect of dependencies on defects in subcomponents. We found that using dependencies to predict defects has no potential benefit. From this, we might prematurely conclude that dependencies should not be used.

As shown in the middle of Figure 1, there are many levels of artifacts. The lowest level in our example is the source file such as Java or C++ source and header files. The higher levels are aggregation of the lower layers. The next level in our example are the subcomponents. For example, the JDT has Java packages that correspond to the Java editor or the debugger. The top level is the software project, which usually corresponds to a software product such as the Eclipse project[2]. A software project usually contains components. Each component is an aggregation of subcomponents, e.g., Java packages. For example, Eclipse consists of the Platform core, the Java development tools (JDT), the Plug-in Development Environment (PDE), and others. The levels in Figure 1 are only an example. Source files can be thought as aggregations of methods. Methods can be thought as aggregations of statements. The distinction depends on the software projects and/or the interpretation of the researchers.

Studies usually use one level of artifact implicitly. For example, Zimmermann et. al. [28] studied the benefits of using dependency structure in defect prediction at the component level. They found that dependencies improve the prediction. The implicit assumption here is that the study findings should be reflected equally at all levels of artifacts.

[2]The Eclipse Foundation: http://www.eclipse.org

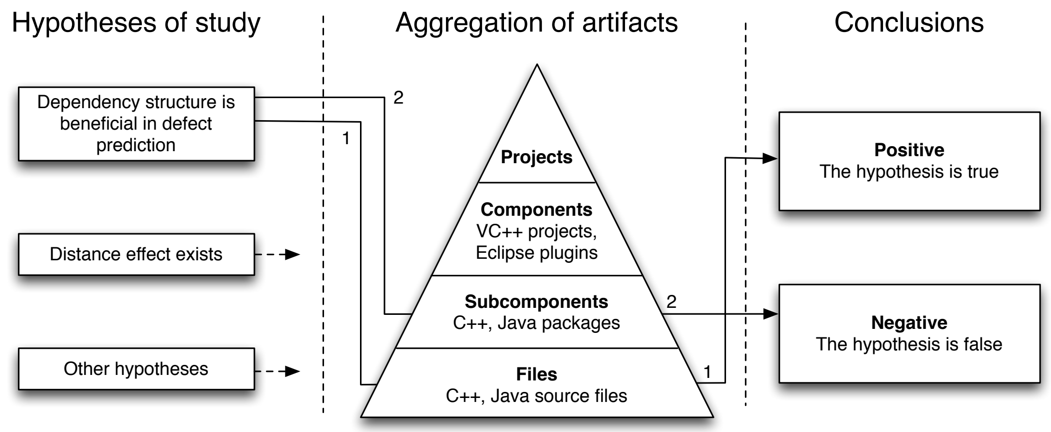

Fig. 1. A simplified view of studies that use software artifacts.

However, researchers have been showing evidences of various types of bias in software artifacts that can cause contradicting findings in studies: erroneous tags [14], [18], missing links [16], [18], [22], incompleteness [15], or aggregation bias [17], [19], [20], [21]. We suspect that a particular kind of bias, which we call aggregation bias, is the cause of the contradicting findings on the distance effect. Aggregation bias happens when one evaluates the same hypothesis using a software artifact (e.g., Java classes) and aggregation of those artifacts (e.g., Java packages), and the findings contradict. Empirical evidence of aggregation bias is reported in many studies [17], [19], [20], [21].

In our previous study, we came across evidence of aggregation bias in defect prediction. We replicated Zimmermann et. al.'s study on using social network analysis metrics to predict defects [28]. Our replication study [19] used two levels of artifacts. We found that the original findings hold at the file level (path 1 on Figure 1) but not at the subcomponent level (path 2 on Figure 1). Kamei et. al. [17] also found the same contradicting findings at Java file and package levels using other metrics for defect prediction. Posnett et. al. [20], [21] found that, in defect prediction, analyzing different level of artifacts yield different findings. More recent results on defect prediction also found different accuracies on different level of artifacts [29].

### *C. Research Questions*

Similar to defect prediction, it is possible that the different levels of artifacts are also the reason why we see conflicting findings on the distance effect. Hence, in this study, we study the distance effect in one global software project at two different levels of artifact: source files and components. If the distance effect exists at both levels, we can confirm the findings of Herbsleb et. al. [8] and Cataldo et al. [5], [6], [7]. If the distance effect does not exist at neither levels, we can confirm the findings of our previous study [11] and Bird et. al. [10]'s. If the distance effect exists at one level but not at the other, the conflicting evidence is probably caused by the different levels of artifacts.

To guide our research, we address the following research questions:

- **RQ1: Does the distance effect exist at the source file level?**
- **RQ2: Does the distance effect exist at the component level?**

We answer these research questions in Section IV and Section V, respectively.

## III. Methodology

This section discusses our data collection technique. We will also introduce the research constructs for distance and quality. Finally, we explain the statistical analyses that we will use.

### *A. Data Collection*

In order to study the distance effect, we need to study a large distributed development project. We choose IBM Rational's Jazz project. The Jazz project aims to build a development environment that tightly integrates programming, communication, and project management. The project development involves about 200 project members on 16 different sites that are located in the USA, Canada, and Europe. The team follows an agile development methodology. Each iteration takes about 8-12 weeks and may result in one or two deliveries, either in the middle and/or at the end of the iteration. We have access to data of five iterations, which span about 14 months from 2007 to 2008.

What is unique about the Jazz project is that the project maintains a very clean repository of artifacts, since it is also the team's product. The artifacts are linked properly, i.e., we can connect developers, whose work locations are known, to the files they modified. All the artifacts are also marked properly. Hence, we can distinguish if a code change is a bug fix or an enhancement. We may be able to use heuristics to mine the same information from open-source projects. However, the errors of the heuristics might introduce linkage biases [16], [18], which we want to avoid in this study.

There are 47,669 work items in the repository. We perform three filtering steps to eliminate invalid artifacts from our study. First, we remove work items that were not resolved by a change to the code. A large part of these work items are the result of migrating from a legacy system where the system was developed initially. Second, we remove all the non-source related files such as machine generated, build, or configuration files. Third, there are cases when a developer has to fix a defect he/she found, but for which no work item exists yet. Typically, he/she would perform the fix first. Then he/she creates a work item for it. Otherwise, he/she cannot check in the code. In that case, the resolution time of the work item does not reflect the actual resolution time of the defect. It will only be a few minutes. So, we exclude these work items from our analysis by removing work items that (a) have only one contributor, i.e., authors, subscribers, or commenters; and (b) are closed within one hour. After these three steps, we have 15,924 work items that are valid for our study.

The cleaned data contains 21,434 source files in 88 components. We divide the artifacts into two categories [30]: distributed and collocated. At the file level, the categories are determined by the number of separate work sites, from which team members have modified the file across time. If the contributors of a file are from more than one site, the file is a distributed source file. If all contributors are from the same location, it is a collocated file. There are 1,856 distributed files and 19,578 collocated files. We use a similar classification for components. The number of distributed and collocated components are 31 and 57 respectively.

## B. Measures of Software Productivity and Quality

Our goal is to determine the effect of distance using software artifacts. As in previous studies [10], [5], [6], [7], [8], [31], [28], we define the following two constructs as measures of software productivity and quality: resolution time and defect count.

*1) Resolution time:* The resolution time is the time to fix a defect or to implement an enhancement. When a defect is discovered or an enhancement is planned, a work item is created. When the defect is fixed or the enhancement is implemented, the work item will be closed. Similar to Herbsleb et. al. [8], we take the difference between the two timestamps as the resolution time. Following the ITLT V3 standard [32], the resolution time of a component is calculated as the average resolution time of all defects and enhancements in that component. The resolution time of a source file is the average resolution time of all defects and enhancements that modified the source file. The smaller the resolution time, the more defects and enhancements can be completed in a shorter time, thus, the higher the productivity. We note that the studied work items were worked on and completed within the studied five milestones. Since, unlike other datasets, the work items only rarely are de-prioritized and moved from one iteration to another, the resolution time is a good estimate of the time required to complete the work item.

*2) Defect count:* The defect count of a source file or a component is the number of defects that is associated with the file or the component. We only count the number of work items, associated with the artifacts, that are marked as a defect. The higher the count, the more defect-prone the file or component is. Software quality studies usually use the defect count as construct for software quality [31], [28]. Bird et. al. [10] and Cataldo et. al. [5], [6], [7] used defect count on software components in their studies on the distance effect.

## C. Confounding factors

To study the existence of the distance effect, we compare the aforementioned measures between distributed and collocated artifacts. However, the cause of the differences may not be the distance. It is possible that distributed or collocated artifacts share a common property that makes them harder to patch and more prone to defects. If this is the case, the relationship between distance and software productivity and quality is just spurious. Thus we have to eliminate plausible confounding factors.

Based on prior research [33], [34], [35], [10], we identify three possible confounding factors: the number of changes, the code size, and the team size. Change typically introduces new bugs [33], [34]. Perhaps distributed or collocated artifacts are changed more often than the other. Furthermore, the larger the files, the higher the number of lines of code they contain. If defect density is distributed evenly per lines of code, then larger artifacts are more prone to defects [35]. Finally, components developed by a larger team are more subjective to defects [10] since the more developers, the higher the possibility of defect occurrences, independent of whether or not they are distributed.

## D. Statistical Analyses

Table II summarizes the statistical analyses that we use to answer RQ1 and RQ2. There are two levels of artifacts: source file and component. For each level, we perform: 1) a straight comparison on the quality measures between distributed and collocated artifacts; 2) comparison that is controlled for confounding factors, one at a time. The controlled comparison will tell if the difference in the straight comparison is indeed caused by the distance effect or by the confounding factor; and 3) a multivariate analysis is used to validate the results of the controlled comparisons, by controlling for all confounding factors at once.

*1) Comparison of distributions:* To compare resolution time and defect count between distributed and collocated artifacts, we use box plots and the Mann-Whitney-Wilcoxon (MWW) test [36].

Box plots are commonly used in exploratory data analysis. The middle line of the plot represents the $50^{th}$ percentile, which is the median of all values. The box represents the $25^{th}$ and the $75^{th}$ percentile. The whiskers show the minimum and the maximum observations, excluding outliers. We can compare the quality measures between distributed and collocated artifacts by comparing the box plot side by side. For confidentiality reasons, we will not display the linear scale on the y-axis. This, however, should not affect our result since we are only interested in the comparison between distributed and collocated artifacts.

Observations on the box plots alone can be deceiving. Hence, we need a statistical test that can tell us if the difference is statistically significant. Since both resolution time and defect

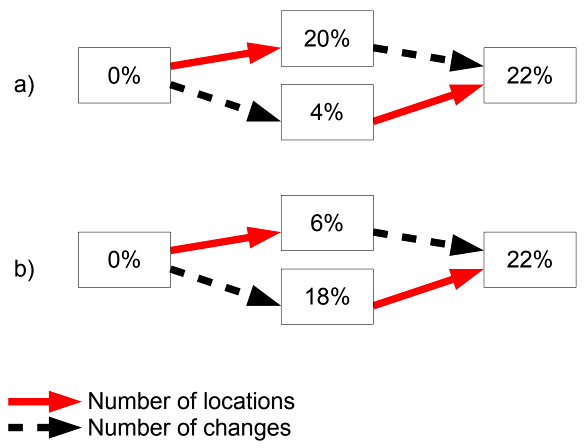

Fig. 2. Example of a multivariate analysis. In a), most of the variance is explained by the distance effect. The confounding factor, i.e., the number of changes has only a small effect. So, the distance effect is the main factor. In b), the confounding factor is the main factor.

counts are not normally distributed, we use the non-parametric MWW test [36]. The null hypothesis of the MWW test is that there is no difference between the two groups. We pick a common level of significance of $\alpha = 0.05$. If the test outputs a $p$ value smaller than 0.05, we reject the null hypothesis, which means that there is most likely a significant difference between the two groups.

*2) Controlled comparisons:* At each level of analysis, we will first compare the distributed and collocated artifacts' quality measures. If the difference is significant, we can say that the distance effect exists. Then, for each possible confounding factor, we perform a controlled comparison. We apply equal frequency discretization on the confounding factor, i.e., we divide the artifacts into equal-size classes according to the confounding factor (e.g., the 50% classes with many changes and 50% classes with few changes). Then, we compare the difference of quality between the distributed and the collocated artifacts within each class. This way, the effect of the confounding factor is minimized. If the difference still exists within the classes, we can say that distance has a major effect on software productivity and quality. Otherwise, the confounding factor plays a major role instead.

*3) Multivariate Analysis:* To confirm the results of the comparisons by controlling for all confounding factors at once, we use multivariate analysis. For this analysis, we build generalized linear models (GLM) that predict the quality measures using the distance measure and the confounding factors. For each model, we add one factor after the other. At each state, we compute the statistical deviance of the model from a null model. The higher the deviance explained, the more the model explains the variance in the quality constructs. Instead of reporting the models's results separately, which would be hard to read, we create a visualization of all the models together. Using the visualization, we can observe the increase in deviance explained after each state to judge whether the distance measure or one of the confounding factors contributes the most to the variance in the quality construct.

Figure 2 shows two examples of the multivariate analysis used in this study. Figure 2a is an example where the distance effect is the main factor. Figure 2b is an example where the confounding factor is the main factor. The graph is a summary of all the models in the analysis. Each path from the first node to the last node corresponds to a model. The nodes show the deviance explained at each step of the construction. The first node is the null model, so the deviance explained is 0%. The last node is the full model when all the factors are used, so the deviance explained is the maximum we can get from all the factors (22%). The arrows show which factor is applied at each step. For example, in the upper path, we first add the distance measure (the solid red arrow), then the confounding factor (the dashed black arrow). We do the opposite for the lower path.

In Figure 2a, when we add the distance measure into the model (on the upper path), the deviance explained (20%) increases to almost the maximum (22%). Adding the confounding factor only increases by 2%. On the lower path, when we add the confounding factor first, the deviance explained only increases by 4%. The distance measure contributes to the remaining deviance. Hence, for the example in Figure 2a, the distance measure is the main contributor to variance in the quality construct. Figure 2b shows the opposite situation when the confounding factor is the main factor.

To construct the GLMs for resolution time, we first have to determine the probability distribution and link function [37]. We try to fit both the resolution time and the defect counts using different probability distributions, i.e., Binomial, Gaussian, Inverse Gaussian, Poisson, and Gamma. As in Cataldo et. al.'s study [6], we found that the Binomial distribution fits best. To determine the link function, we build GLMs using the Binomial distribution with different link functions, i.e., logistic, normal, complementary log-log, and Cauchy. We compute the Akaike information criterion (AIC) for each GLM. The lower the AIC, the fitter the model [37]. The result shows that the Cauchy link function works best.

For confidentiality reasons, we scale the maximum deviance explained to 100% of the full model. For example, if the deviance explained is 30% for the full model and 10% for an intermediate state, we will display 100% and 33% respectively on the graph. This scaling does not affect our result since we are only interested in comparing the relative differences in the deviance explained.

## IV. RQ1: Does the distance effect exist at the source file level?

In this section, we determine if distance has an impact on quality of source files by applying the analyses described in Section III-D.

### *A. Comparing distributed and collocated files*

Figure 3 shows the box plots of response time and defect counts of distributed and collocated files. Figure 3a compares the resolution time between distributed and collocated source files for both defects and enhancements. We observe that the box plot of the distributed source file is higher. This means

TABLE II
ANALYSES

| Level | Different Analysis | Construct | Description |
|---|---|---|---|
| File | Comparison | Resolution time and defect count | Distributed vs Collocated |
| | | Resolution time | Controlled for size |
| | | | Controlled for change |
| | Multivariate Analysis | Resolution time | Compare effect of size, change, and distance |
| Component | Comparison | Resolution time and defect count | Distributed vs Collocated |
| | | Defect count* | Controlled for size |
| | | | Controlled for number of people |
| | Multivariate Analysis | Defect count* | Compare effect of size, people, and distance |

(*) The resolution time comparison is not statistically significant so checking for confounding factors is not required.

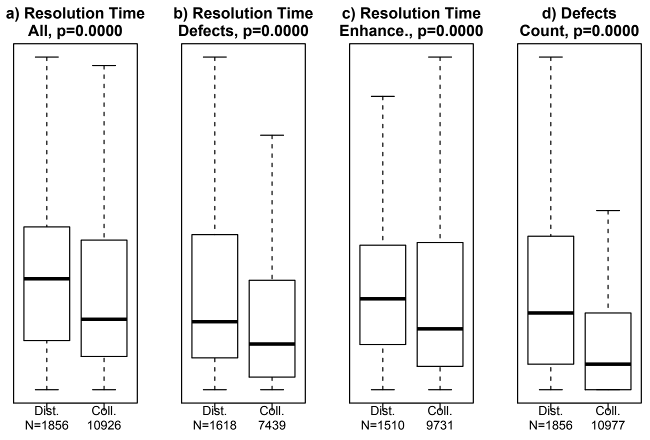

Fig. 3. Comparing the quality between distributed and collocated source files. Resolution time is measured in hours. For confidentiality reasons, we do not display the linear scale on the y-axis of the box plots.

that distributed source files take longer to patch. The MWW test confirms that the difference is statistically significant ($p < 0.05$). We print the $p$ value at the top of each box plot. Figure 3b compares the resolution time of only defects, whereas Figure 3c compares the resolution time of only enhancements. In both cases, distributed source files take longer to patch. The difference is clearer for defects compared to the enhancements. Similarly, Figure 3d shows that distributed files usually contain more defects than collocated files do. We note that the number of files in Figure 3b and Figure 3c do not add up to the number in Figure 3a because there are files that are modified by both defects and enhancements.

## B. Comparing quality with controlled confounding factors

For source files, the two major confounding factors are the number of changes to the code and the code size. The third factor, the team size is only applicable for components. To control for the two confounding factors, we compare the resolution time of distributed and collocated files across evenly distributed classes of number of code changes and code size. If each class exhibits similar differences between the distributed and collocated files as in the straight comparison, we can safely rule out the factor. Otherwise, the confounding factor might be the main factor instead.

Figure 4 shows the box plots of the distributed and collocated files when controlling for the number of changes. We apply equal frequency discretization on the number of changes, i.e., we divide the number of changes into five classes of equal size, each containing 20% of the data. The first one has the smallest number of changes (2 changes). The fifth one has the largest number of changes (10 to 37). The triplet on top of the box plot indicates the minimum, the median, and the maximum number of changes in that class. We calculate the $p$ value of the MWW test between resolution time of distributed and collocated files within each class. We show the $p$ value at the top of each box plot. Interestingly, when the number of changes is small, the collocated files take longer to patch, contradictory to the straight comparison. In the third category, the difference is not statistically significant ($p > 0.05$). In the remaining categories, distributed files take longer to patch. The differences are statistically significant ($p < 0.05$).

Figure 5 shows box plots of the distributed and collocated files when controlling for code size. Similarly, we divide source files into six classes of equal size according to the code size. The first one has the smallest files; the sixth one has the largest files. We also print the $p$ value of the MWW test at the top of each box plot. Interestingly, when the size is very small or very large, there is no difference in resolution time between distributed and collocated files ($p > 0.05$). However, in the size categories in between, distributed files take longer to patch.

Judging by the differences found in both controlled comparisons, we can say that the number of changes plays some role in the difference observed in the straight comparison. For source files with only two changes, collocated files take longer than distributed files, which is contradictory to the straight comparison. The effect of code size, on the other hand, is much smaller. Although the smallest and the largest files exhibit no statistically significant difference in resolution time, the box plots show that the distributed files still take slightly longer to patch. We confirm the role of the two confounding factors using a multivariate analysis in the next subsection.

## C. Multivariate analysis

Figure 6 shows the results of the multivariate analysis (see Section III-D) on the resolution time of source files. For each source file, we consider three possible factors that can affect the resolution time: distance measure, code size, and number of changes. Distance measure is the number of development locations. It represents the distance effect in our analysis, while size and change are the two confounding factors.

The result of this analysis confirms the results of the controlled comparison in the previous subsection. The distance measure has a significant effect on resolution time (71%) by itself. Number of changes also has a significant effect (55%). Size, on the other hand, has a much smaller effect (12%). Hence, the major confounding factor is the number of changes.

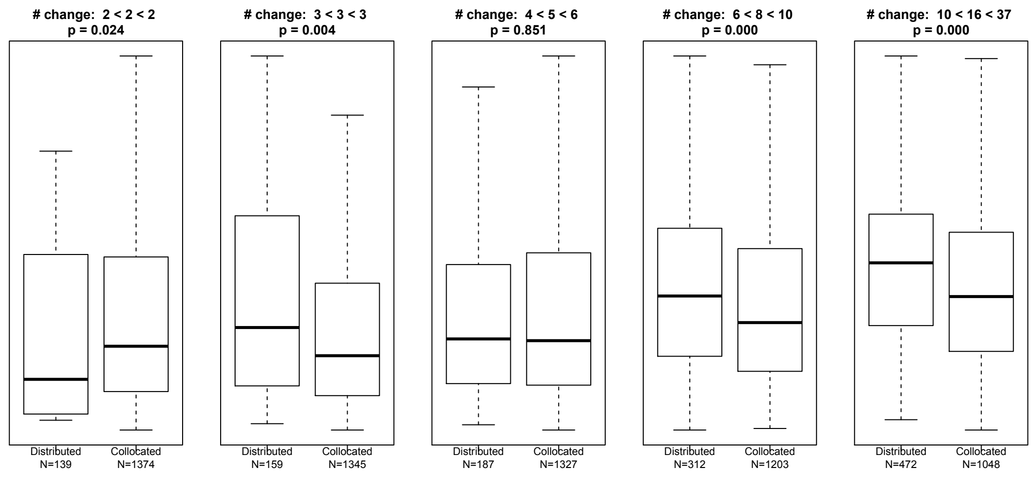

Fig. 4. Comparing the resolution time of distributed and collocated source files controlled by the number of changes.

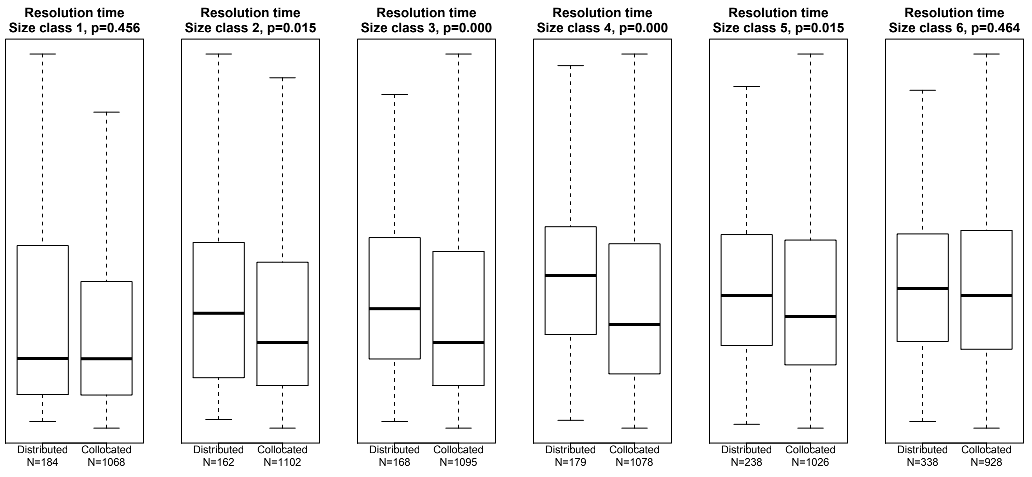

Fig. 5. Comparing the resolution time of distributed and collocated source files controlled by the size of the files.

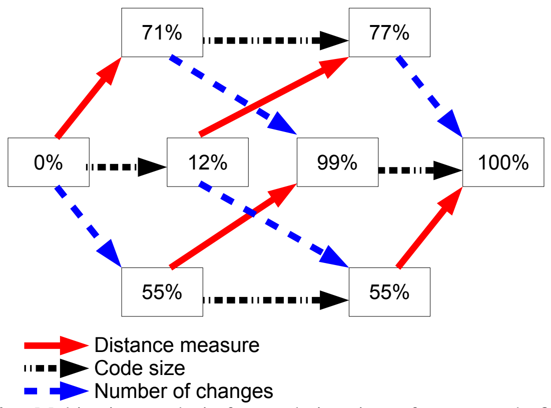

Fig. 6. Multivariate analysis for resolution time of source code files. The distance measure is the number of team locations where the file has been modified (the solid red arrows). The confounding factors are the size of the code (the dashed dotted black arrows) and the number of changes to the code (the dashed blue arrows). For confidentiality reasons, we scale the deviance explained to 100% of the full model.

To compare the effect of distance and the effect of the number of changes, we look at the dashed blue, solid red, then dashed dotted black path. When we add the number of changes to the model (the dashed blue arrow), the deviance explained increases to 55%. Then, when we add the distance measure (the solid red arrow), the deviance explained almost doubles to 99%, which is an additional increase of 44% on top of the 55%. This means that the distance measure contributes to the model as much as the number of changes.

## D. Conclusion

The combined results show that the distance effect negatively impacts the quality of source files. We arrive at this conclusion due to three reasons. The first reason is that the resolution time of distributed files is longer than that of collocated files, based on the straight comparison (Section IV-A). Second, when we perform the comparison controlling for the confounding factors, distributed files are still taking longer to patch in the majority of cases (Section IV-B). This means that the distance effect must have a significant impact on the delay in resolution time. Thirdly, our multivariate analysis confirms the results of the controlled comparisons (Section IV-B). The analysis shows that the distance effect has a larger impact than the two confounding factors.

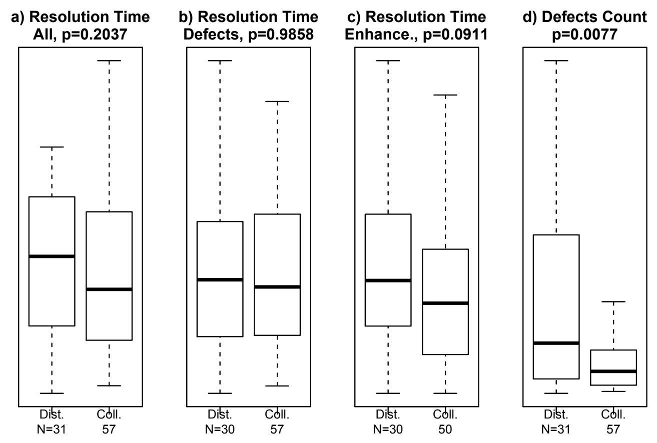

Fig. 7. Comparing the quality of distributed and collocated components. The resolution time of a component is measured in hours.

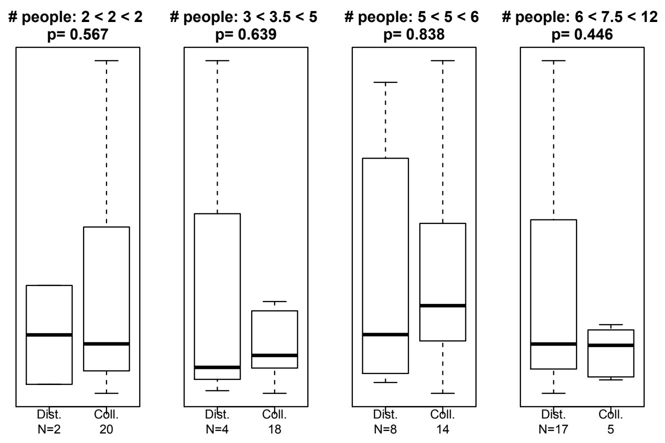

Fig. 9. Comparing the defect counts of distributed and collocated components contro

## V. RQ2: Does the Distance Effect Exist at the Component Level?

In this section, we determine if distance also has an impact on quality of components. Similar to RQ1, we apply the same analyses as described in Section III-D.

### *A. Comparing distributed and collocated components*

The straight comparison between distributed and collocated components is plotted in Figure 7. Figure 7a shows that the average resolution time of distributed components is higher than that of the collocated components. However, a MWW test shows that the difference is statistically insignificant ($p > 0.05$). Similar observations can be made for defects and tasks by themselves, as shown by Figure 7b and Figure 7c, respectively. It seems that the distance effect does not have a significant influence on resolution time at the component level.

However, the defect count shows the opposite. As we can see in Figure 7d, distributed components contain more defects than collocated components (80 vs 35). This difference is statistically significant ($p < 0.05$). Hence, it is possible that the distance effect has a negative impact on the quality of software components. Thus, the controlled comparisons and the multivariate analysis need to find out if the distance effect is the main factor that influences the defect counts.

### *B. Comparing quality with controlled confounding factors*

We identify two confounding factors that might influence the component's number of defects: code size and team size. The third factor, the number of changes, is not meaningful for this analysis because, by definition, it highly correlates with the number of defects.

Figure 8 compares the defect count between the distributed and collocated components when controlling for the total code size of the component. We divide the components into five classes of equal size according to the code size. The first class has the smallest components, whereas the fifth one has the largest components. We print the $p$ value of the MWW test at the top of each box plot. Interestingly, except for the components in the third size class, none of the differences in defect count are statistically significant. The third class shows a significant difference, yet the collocated

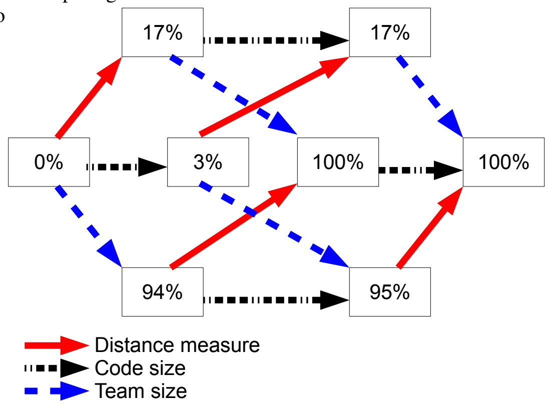

Fig. 10. Multivariate analysis for defect count of components. The distance measure is the number of team members' location (the solid red arrows). The confounding factors are the code size (the dashed dotted back arrows) and team size (the dashed blue arrows).

components have more defects. These results indicate that size is probably a significant confounding factor because the significant difference shown in the straight comparison does not appear within the classes of size.

Figure 9 compares the defect count between distributed and collocated components when controlling for the number of team members. We divide components into four classes of equal size according to the number of contributors. The first class has the lowest number of contributors (2). The fourth has the highest number of contributors (6 to 12). We use four classes instead of five in this case because, if we use five classes, one class does not have any distributed component to compare to. The results show that none of the differences is statistically significant. It means that the number of team members is also a significant confounding factor.

The results of the controlled comparison show that both code size and team size seem to be major confounding factors. Hence, as in RQ1, we perform a multivariate analysis to determine the most influential factor.

### *C. Multivariate analysis*

Figure 10 shows the multivariate analysis (see Section III-D) on the defect count of components. For each component, we consider three possible factors that can affect the defect

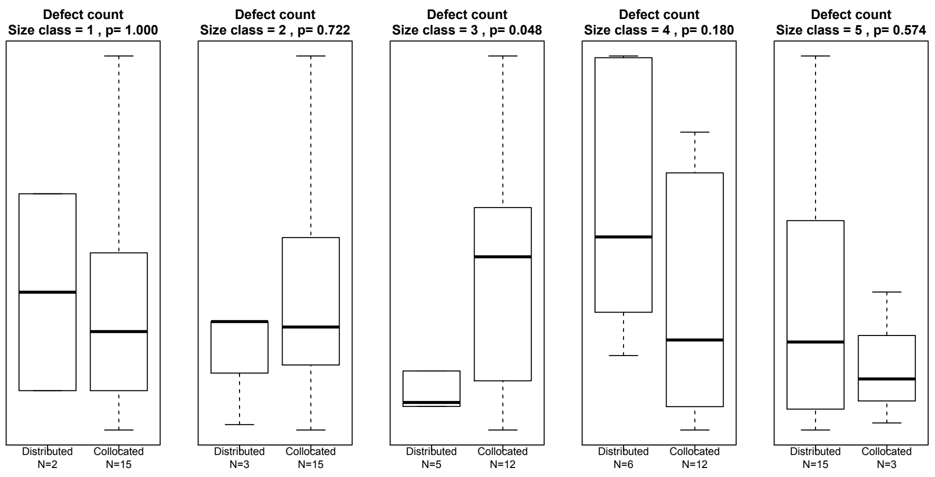

Fig. 8. Comparing the defect counts of distributed and collocated components controlled by the code size of the component.

count: distance measure, code size, and team size. The distance measure is the number of locations. It represents the distance effect in our analysis. Code size and team size, on the other hand, are the two confounding factors.

The result shows that distance has a significant impact on the number of defects (17%). However, the impact of the team size is much larger (94%). The code size does not contribute much. Although team size has a much larger influence, what we want to find out is if the distance effect also contributes to the full model. Hence, we look at the dashed blue, solid red, and then the dashed dotted black path. When we add the distance measure (the solid red arrow) after adding the team size (the dashed blue arrow), the deviance explained increases very little (from 94% to 100%). This implies that most of the distance effect's impact has already been explained by the team size. This means that the difference in defect count between distributed and collocated components is affected mostly by the team size. It is possible that distributed teams simply have more contributors. They were able to locate and fix more defects. So, their components simply have more defects.

### *D. Conclusion*

Our results support the argument that distance has no significant effect on the quality of software components due to three reasons. The first reason is that there is no significant difference in resolution time between distributed and collocated components in the straight comparison (Section V-A). The second reason is that even when there is a significant difference in defect count between distributed and collocate components in the straight comparison, there is no difference in the controlled comparisons (Section V-B). The third reason is that the multivariate analysis (Section V-B) confirms that even though distance can have a significant impact on the defect count, most of the impact can be explained by the team size.

## VI. Discussion

### *A. Can aggregation bias explain the contradictory findings on the distance effect?*

The findings from our study compared to previous studies (see Table I) are:

- **RQ1 (Section IV):** The effect of distance can be observed at the file level which confirms the findings from Herbsleb at. el. [8] and Cataldo at. el. [5], [6], [7].
- **RQ2 (Section V):** The effect of distance cannot be observed at the component level which confirms the findings of Nguyen at. el. [11], Bird at. el. [10], and Spinellis [12].

Since there are strong evidences of aggregation bias in defect prediction [19], [17], [20], [21], we would argue that the results indicate that aggregation bias might play a role in the contradictory findings.

### *B. Why does aggregation bias exist?*

Aggregation bias has been a debate since the early 1950s [23]. There are many examples that demonstrate the causes of aggregation bias in various subjects. For example, Robinson [38] showed that, according to the United States 1930 census, new immigrants are more likely to be illiterate. However, states that have more immigrants are less illiterate. So it would not be logical to make inference about individuals from properties of the aggregate. In the case of immigrant and illiteracy, the cause of aggregation bias is that new immigrants tend to settle in states that have less illiteracy.

Unfortunately, we cannot provide the causes for aggregation bias in our study. One possibility is that the different levels of artifacts represent different socio-psychological processes that are not linearly related. Studies in sociology found that the linkage among units of an organization can cause the system to exhibit effects that are different from that of a collection of individual units [39]. One mechanism of organization linkages is called the "negative feedback loop". When a unit of an organization exhibits a negative effect on the organization, other units compensate. So, even though there is a negative effect at the unit level, at the organization level, there is no negative effect. For our case study, it is possible that the difficulty in making changes to the distributed source files urges the developers to be more efficient in other activities, such as integration testing or planning. Such compensation might have cancelled out the negative effect of distance. So, when we look at the components, there is no noticeable effect.

### C. *What does this mean for past and future research?*

The existence of aggregation bias poses the following challenge in the use of software artifacts to study software quality and processes: Every software artifact is an aggregation of another one. One would argue that a class file is actually aggregation of the functions in the class. The functions themselves consist of statements. Thus, we cannot even conclude that the distance effect exists in general using just the data from software artifacts in our case study, i.e., the Jazz project.

Because of that challenge, future research should:

**Independently verify findings using other research methodology such as surveys, interviews, or ethnography:** We were eventually able to talk to the Jazz team members. They did confirm informally that they encounter serious challenges with distributed development. Reallocation of team members was required in some instances. The challenge of distance is also prevalent in industry's sources. For example, one of the Agile Manifesto's principle is: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation [40].

**Replicate many of the past studies to see if there are missing results:** Many studies that use software artifacts only study a single level of artifacts. So, researchers should be encouraged to conduct replication studies using other artifacts. The fact that a study has established a relationship between a quality construct and properties of certain software artifacts should not discourage other researchers to examine the same relationship using other levels of artifacts. Researchers can also explore other levels of artifacts. For example, recent studies [41], [42], [43] suggested that defect prediction should be done on metrics of a lower level artifact than class file: the change itself.

## VII. Threats to Validity

**Internal validity.** We checked three possible confounding factors: file size, number of changes, and team size. There are, however, other possible confounding factors such as complexity [31] or task dependencies [7].

We use the same distance construct as prior studies: collocated vs distributed. However, there are newer and finer-grain constructs for distance [7], [44] that can also be considered.

The MWW test requires that samples are independent. This is true for components. However, there are work items that deliver changes to both distributed and collocated files. For this reason, we confirmed our findings with a multivariate analysis, which found that distance has a major effect on the resolution time. We also ran an ANOVA analysis on the final model. It shows that both the distance measure and the number of changes are statistically significant, while the code size is not. Hence, we can safely say that the distance effect has a real impact on resolution time of distributed source files.

When there are tests on overlapping data, a stronger level of confidence must be used. In most cases, when we apply multiple MWW tests, the data are not overlap. For example, in the file level comparisons controlling for the number of changes (Figure 4), the files in each class are separated. However, the straight comparison at the file level (Figure 3 a, b, and c) has overlapping data because some files are changed by both defect fixes and enhancements. We apply the Bonferroni correction [37] for these three hypothesis tests. The correction essentially divides the $p$ value of each test by three. The results still hold. All the differences shown on Figure 3 are still statistically significant after the correction.

It is possible that the reason why the differences are significant at the file level but not at the component level is because there are more files than components. To analyze this hypothesis, at the file level, we measure the effect size using Cliff-Delta and we find that our results hold with a non-negligible effect size. At the component level, even if the statistics in Figure 7 were significant, our multivariate analysis on defect count in Figure 10 shows that the team size is a significant confounding factor. The same analysis but on the response time shows a similar pattern. So the sample size is not the reason for the contradictory results at the file level and the component level.

The MWW test is also sensitive to an unbalanced number of data points. It may yield false statistical significant results. There are more collocated than distributed artifacts. So, we rerun the tests in RQ1 where the statistical significant results are important to the finding. This time, we randomly sampled the collocated files so the number is the same as the distributed files. We rerun the tests 50 times to eliminate sampling bias. All the tests show similar results except for the first class of number of changes in Figure 4, which becomes statistically insignificant. This does not affect our conclusion for RQ1.

**External validity.** This paper is on improving external validity of studies that use software artifacts. There are many studies that confirmed the existence of aggregation bias on defect counts [17], [19], [20], [21]. This study shows that aggregation bias also affects resolution time. It is possible, however, that other software quality metrics might have more resistance to aggregation bias.

## VIII. Conclusion

In conclusion, we demonstrate that aggregation bias is a possible explanation for the contradictory findings on the effect of distance on distributed teams. We conduct a study of the same software project using two different levels of artifacts. The findings are different, which calls for special attention to the level of artifacts used in studies of software artifacts.

The finding suggests an extra dimension for external replication of past studies. For example, there are many studies that use different product and process metrics [33], [34], [31], [45] to predict different aspects of software quality. Do these approaches work at all levels of artifacts? Such replication would provide a better understanding of defect prediction models.

## Acknowledgment

We are grateful to all members of the IBM Jazz project who helped us during the study, in particular, Harold Ossher, Jean-Michel Lemieux, Li-te Cheng, Kate Ehrlich, and Tim Klinger.

## References

[1] R. D. Battin and R. Crocker, "Leveraging resources in global software development," *IEEE Software*, vol. 18, no. 2, p. 8p, 2001.

[2] E. Carmel, *Global software teams: collaborating across borders and time zones*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

[3] R. E. Grinter, J. D. Herbsleb, and D. E. Perry, "The geography of coordination: dealing with distance in research and development work," in *Proceeding of the International Conference on Supporting Group Work (GROUP)*. ACM, 1999, pp. 306–315.

[4] R. Prikladnicki, J. L. N. Audy, D. Damian, and T. C. de Oliveira, "Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring," in *Proceeding of the ICGSE*, 2007, pp. 262–274.

[5] M. Cataldo, "Sources of errors in distributed development projects: implications for collaborative tools," in *Proceeding of the Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2009, pp. 281–290.

[6] M. Cataldo and S. Nambiar, "On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality," in *Proceeding of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2009.

[7] ——, "Quality in global software development projects: A closer look at the role of distribution," in *Proceeding of the International Conference on Global Software Engineering (ICGSE)*. IEEE Computer Society, 2009, pp. 163–172.

[8] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering (TSE)*, vol. 29, no. 6, pp. 481–495, 2003.

[9] P. Wagstrom and S. Datta, "Does latitude hurt while longitude kills? geographical and temporal separation in a large scale software development project," in *Proceeding of the International Conference on Software Engineering (ICSE)*. ACM, 2014.

[10] 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 *Proceeding of the International Conference on Software Engineering (ICSE)*, 2009.

[11] T. Nguyen, T. Wolf, and D. Damian, "Global software development and delay: Does distance still matter?" in *Proceeding of the ICGSE*. IEEE Computer Society, 2008, pp. 45–54.

[12] D. Spinellis, "Global software development in the freebsd project," in *Proceeding of the International Workshop on Global Software Development for the Practitioner (GSD)*. ACM, 2006, pp. 73–79.

[13] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software*, vol. 18, no. 2, pp. 22–29, 2001.

[14] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y.-G. Guhneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceeding of the Conference of the Center for Advance Studies on Collaboration Research (CASCON)*. Ontario, Canada: ACM, 2008, pp. 304–318.

[15] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceeding of the International Conference on Software Engineering (ICSE)*, Vancouver, BC, 2009.

[16] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *Proceeding of the ESEC/FSE*, 2009, pp. 121–130.

[17] Y. Kamei, S. Matsumoto, A. Monden, K. i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *Proceeding of International Conference on Software Maintenance (ICSM)*, 2010, pp. 1–10.

[18] T. H. D. Nguyen, B. Adams, and A. E. Hassan, "A case study of bias in bug-fix datasets," in *Proceeding of Working Conference on Reverse Engineering (WCRE)*, Beverly, Massachusetts, 2010.

[19] ——, "Studying the impact of dependency network measures on software quality," in *Proceeding of the ICSM*, 2010, pp. 1–10.

[20] D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Proceeding of the International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 2011.

[21] D. Posnett, R. D. Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *Proceeding of the International Conference on Software Engineering (ICSE)*. IEEE Press, 2013.

[22] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. Gall, "Discovering loners and phantoms in commit and issue data," in *Proceeding of the International Conference on Program Comprehension (ICPC)*. IEEE Press, 2015.

[23] P. A. Jargowsky, "The ecological fallacy," *Encyclopedia of social measurement*, vol. 1, pp. 715–722, 2005.

[24] G. M. Olson and J. S. Olson, "Distance matters," *Human-Computer Interaction*, vol. 15, no. 2, pp. 139 – 178, 2000.

[25] P. Hinds and M. Mortensen, "Understanding conflict in geographically distributed teams: An empirical investigation," *Organization Science*, vol. 16, pp. 290–307, 2005.

[26] J. A. Espinosa, N. Ning, and E. Carmel, "Do gradations of time zone separation make a difference in performance? a first laboratory study," in *Proceeding of the ICGSE*, 2007, pp. 12–22.

[27] N. Ramasubbu, M. Cataldo, R. K. Balan, and J. D. Herbsleb, "Configuring global software teams: a multi-company analysis of project productivity, quality, and profits," in *Proceeding of the International Conference on Software Engineering (ICSE)*. ACM, 2012, pp. 261–270.

[28] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceeding of the International Conference on Software Engineering (ICSE)*. ACM, 2008, pp. 531–540.

[29] H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," in *Proceeding of the International Symposium on Software Reliability Engineering (ISSRE)*, 2014, pp. 312–322.

[30] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," in *Proceeding of the Conference on Computer Supported Cooperative Work (CSCW)*. Philadelphia, Pennsylvania, United States: ACM, 2000, pp. 319–328.

[31] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceeding of the International Conference on Software Engineering (ICSE)*. ACM, 2006, pp. 452–461.

[32] Office of Government Commerce, "Information technology infrastructure library (itil v3)," 2007.

[33] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceeding of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2009, pp. 78–88.

[34] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceeding of the International Conference on Software Engineering (ICSE)*. ACM, 2005, pp. 284–292.

[35] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[36] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[37] J. Rawlings, S. Pantula, and D. Dickey, *Applied regression analysis: a research tool*. Springer, 1998.

[38] W. S. Robinson, "Ecological correlations and the behavior of individuals," *American Sociological Review*, vol. 15, no. 3, pp. 351–357, 1950.

[39] P. Goodman, *Missing Organizational Linkages: Tools for Cross-Level Research*. SAGE Publications, 2000.

[40] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mallor, K. Shwaber, and J. Sutherland, "The agile manifesto," Report, 2001.

[41] A. Tarvo, N. Nagappan, and T. Zimmermann, "Predicting risk of pre-release code changes with checkinmentor," in *Proceeding of the International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 128–137.

[42] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: analytics and risk prediction of software commits," in *Proceeding of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 966–969.

[43] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[44] A. R. da Rosa Techio, R. Prikladnicki, and S. Marczak, “Reporting empirical evidence in distributed software development: An extended taxonomy,” in *Proceeding of the International Conference on Global Software Engineering*, 2015, pp. 71–80.

[45] H. Hu, H. Zhang, J. Xuan, and W. Sun, “Effective bug triage based on historical bug-fix information,” in *Proceeding of the International Symposium on Software Reliability Engineering (ISSRE)*, 2014, pp. 122–132.