Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista

Christian Bird[1], Nachiappan Nagappan[2], Premkumar Devanbu[1], Harald Gall[3], Brendan Murphy[2]

[1]University of California, Davis, USA
[2]Microsoft Research
[3]University of Zurich, Switzerland

{cabird,ptdevanbu}@ucdavis.edu {nachin,bmurphy}@microsoft.com gall@ifi.uzh.ch

Abstract

It is widely believed that distributed software development is riskier and more challenging than collocated development. Prior literature on distributed development in software engineering and other fields discuss various challenges, including cultural barriers, expertise transfer difficulties, and communication and coordination overhead. We evaluate this conventional belief by examining the overall development of Windows Vista and comparing the post-release failures of components that were developed in a distributed fashion with those that were developed by collocated teams. We found a negligible difference in failures. This difference becomes even less significant when controlling for the number of developers working on a binary. We also examine component characteristics such as code churn, complexity, dependency information, and test code coverage and find very little difference between distributed and collocated components. Further, we examine the software process and phenomena that occurred during the Vista development cycle and present ways in which the development process utilized may be insensitive to geography by mitigating the difficulties introduced in prior work in this area.

1. Introduction

Globally distributed software development is an increasingly common strategic response to issues such as skill set availability, acquisitions, government restrictions, increased code size, cost and complexity, and other resource constraints [5, 10]. In this paper, we examine development that is globally distributed, but completely within Microsoft. This style of *global development* within a single company is to be contrasted with *outsourcing* which involves multiple companies. It is widely believed that distributed collaboration imposes many challenges not inherent in collocated teams such as delayed feedback, restricted communication, less shared project awareness, difficulty of synchronous communication, inconsistent development and build environments, lack of trust and confidence between sites, etc. [22]. While there are studies that have examined the delay associated with distributed development and the direct causes for them [12], there is a dearth of empirical studies that focus on the effect of distributed development on software quality in terms of post-release failures.

In this paper, we use historical development data from the implementation of Windows Vista along with post-release failure information to empirically evaluate the hypothesis that globally distributed software development leads to more failures. We focus on post-release failures at the level of individual executables and libraries (which we refer to as binaries) shipped as part of the operating system and use the IEEE definition of a failure as "the inability of a system of component to perform its required functions within specified performance requirements" [16].

Using geographical and commit data for the developers that worked on Vista, we divide the binaries produced into those developed by distributed and collocated teams and examine the distribution of post-release failures in both populations. Binaries are classified as developed in a distributed manner if at least 25% of the commits came from locations other than where binary's owner resides. We find that there is a small increase in the number of failures of binaries written by distributed teams (hereafter referred to as distributed binaries) over those written by collocated teams (collocated binaries). However, when controlling for team size, the difference becomes negligible. In order to see if only smaller, less complex, or less critical binaries are chosen for distributed development (which could explain why distributed binaries have approximately the same number of failures), we examined many properties, but found no difference between distributed and collocated binaries. We present our methods and findings in this paper.

In section 2 we discuss the motivation and background

of this work and present a theory of distributed development in the context of the Windows Vista development process. Section 3 summarizes related work, including prior empirical quantitative, qualitative, and case studies as well as theoretical papers. An explanation of the data and analysis as well as the quantitative results of this analysis is presented in section 4. We discuss these results, compare with prior work, and give possible explanations for them in section 5. Finally, we present the threats to the validity of our study in section 6 and conclude our paper with further avenues of study in section 7.

2. Motivation and Contributions

Distributed software development is a general concept that can be operationalized in various ways. Development may be distributed along many types of dimensions and have various distinctive characteristics [9]. There are key questions that should be clarified when discussing a distributed software project. Who or what is distributed and at what level? Are people or the artifacts distributed? Are people dispersed individually or dispersed in groups?

In addition, it is important to consider the way that developers and other entities are distributed. The distribution can be across geographical, organizational, temporal, and stakeholder boundaries [15]. A scenario involving one company outsourcing work to another will certainly differ from multiple teams working within the same company. A recent special issue of IEEE Software focused on globally distributed development, but the majority of the papers dealt with offshoring relationships between separate companies and outsourcing, which is likely very different from distributed sites within the same company [2, 6, 7]. Even within a company, the development may or may not span organizational structure at different levels. Do geographical locations span the globe, including multiple time zones, languages, and cultures or are they simply in different cities of the same state or nation?

We are interested in studying the effect of globally distributed software development within the same company because there are many issues involved in outsourcing that are independent of geographical distribution (e.g. expertise finding, different process and an asymmetric relationship). Our main motivation is in confirming or refuting the notion that global software development leads to more failures within the context of our setting.

To our knowledge, this is the first large scale distributed development study of its kind. This study augments the current body of knowledge and differs from prior studies by making the following contributions:

1. We examine distributed development at multiple levels of separation (building, campus, continent, etc.).
2. We examine a very large scale software development effort, composed of over 4,000 binaries and nearly 3,000 developers.
3. We examine many complexity and maintenance characteristics of the distributed and collocated binaries to check for inherent differences that might influence post-release quality.
4. Our study examines a project in which all sites involved are part of the same company and have been using the same process and tools for years.

There is a large body of theory describing the difficulties inherent in distributed development. We summarize them here.

Communication suffers due to a lack of unintended and informal meetings [11]. Engineers do not get to know each other on a personal basis. Synchronous communication becomes less common due to time zone and language barriers. Even when communication is synchronous, the communication channels, such as conference calls or instant messaging are less rich than face to face and collocated group meetings. Developers may take longer to solve problems because they lack the ability to step into a neighboring office to ask for help. They may not even know the correct person to contact at a remote site.

Coordination breakdowns occur due to this lack of communication and lower levels of group awareness [4, 1]. When managers must manage across large distances, it becomes more difficult to stay aware of each person's task and how they are interrelated. Different sites often use different tools and processes which can also make coordinating between sites difficult.

Diversity in operating environments may cause management problems [1]. Often there are relationships between the organization doing development and external entities such as governments and third party vendors. In a geographically dispersed project, these entities will be different based on location (e.g. national policies on labor practices may differ between the U.S. and India).

Distance can reduce team cohesion in groups collaborating remotely [22]. Eating, sharing an office, or working late together to meet a deadline all contribute to a feeling of being part of a team. These opportunities are diminished by distance.

Organizational and national cultural barriers may complicate globally distributed work [5]. Coworkers must be aware of cultural differences in communication behaviors. One example of a cultural difference within Microsoft became apparent when a large company meeting was originally (and unknowingly) planned on a major national holiday for one of the sites involved.

Based on these prior observations and an examination of the hurdles involved in globally distributed development we expect that difficulties in communication and coordination will lead to an increase in the number of failures in code pro-

duce by distributed teams over code from collocated teams. We formulate our testable hypothesis formally.

H1: Binaries that are developed by teams of engineers that are distributed will have more post-release failures than those developed by collocated engineers.

We are also interested to see if the binaries that are distributed differ from the collocated counterparts in any significant ways. It is possible that managers, aware of the difficulties mentioned above, may choose to develop simpler, less frequently changing, or less critical software in a distributed fashion. We therefore present our second hypothesis.

H2: Binaries that are distributed will be less complex, experience less code churn, and have fewer dependencies than collocated binaries.

3. Related Work

There is a wealth of literature in the area of globally distributed software development. It has been the focus of multiple special issues of IEEE Software, workshops at ICSE and the International Conference on Global Software Engineering. Here we survey important work in the area, including both studies and theory of globally distributed work in software development.

There have been a number of experience reports for globally distributed software development projects at various companies including Siemens [14], Alcatel [8], Motorola [1], Lucent [11], and Philips [17].

Effects on bug resolution

In an empirical study of globally distributed software development [12], Herbsleb and Mockus examined the time to resolution of Modification Requests (MRs) in two departments of Lucent working on distinct network elements for a telecommunication system. They found that when an engineer was blocked on a task due to information needs, the average delay was .9 days for same site information transfer and 2.4 days if the need crossed site-boundaries. An MR is classified as "single-site" if all of contributors were resided at one site and "distributed" otherwise. The average time needed to complete an "single-site" MR was 5 days versus 12.7 for "distributed". When controlling for other factors such as number of people working on an MR, how diffused the changes are across the code base, size of the change, and severity, the effect of being distributed was no longer significant. They hypothesize that large and/or multi-module changes are both more time consuming and more likely to involve multiple sites. In addition, these changes require more people, which introduce delay. They conclude that distributed development indirectly introduces delay due to correlated factors such as team size and breadth of changes required.

They also introduce practices that may mitigate the risks of distributed development. These include better communication via instant messaging and group chat rather than telephones, better ways of identifying experts to ask for information, and shared calendars and other presence awareness tools to facilitate contact with remote colleagues.

Panjer *et al* [23] observed a portion of a geographically distributed team of developers for two months in a commercial setting and interviewed seven developers from the team. Developers reported difficulty coordinating with distributed coworkers and sometimes assigned MRs based on location. They also created explicit relationships between MRs to capture and manage technical information.

Thanh *et al* [21] examined the effect of distributed development on delay between communications and time to resolution of work items in the IBM's Jazz project, which was developed by developers at five globally distributed sites. They categorized work items based on the number of distributed sites that contributed to their resolution and examined the mean and median time to resolution and time between comments on each work item. While Kruskal-Wallis tests showed a statistically significant difference in the times for items that were more distributed, the Kendall Tau correlations of time to resolution and time between comments with number of sites was extremely low (below 0.1 in both cases). This indicates that the effect of distributed collaboration does not have a strong effect.

In [13], Herbsleb and Mockus formulate an empirical theory of coordination in software engineering and test hypotheses based on this theory. They precisely define software engineering as requiring a sequence of choices for all of the decisions associated with a project. Each decision constrains the project and future decisions in some way until all choices have been made and the final product does or does not satisfy the requirements. It is therefore important that only feasible decisions (those which will lead to a project that does satisfy the requirements) be made. The presented theory is used to develop testable hypotheses regarding productivity, measured as number of MRs resolved per unit time. They find that people who are assigned work from many sources have lower productivity and that MRs that require work in multiple modules have a longer cycle time than those which require changes to just one.

Unlike the above papers, our study focuses on the effect of distributed development on defect *occurrence*, rather than on defect *resolution time*.

Effects on quality and productivity

Diomidis Spinellis examined the effect of distributed development on productivity, code style, and defect density in the FreeBSD code base [25]. He measured the geographical distance between developers, the number of defects per source file, as well as productivity in terms of number of lines committed per month. A correlation analysis showed that there is little, if any, relationship between geographic distance of developers and productivity and defect density. It should be noted that this is a study of open source soft-

ware which is, by its very nature, distributed and has a very different process model from commercial software.

Cusick and Prasad [6] examined the practices used by Wolters Kluwer Corporate Legal Services when outsourcing software development tasks and present their model for deciding if a project is offshorable and how to manage it effectively. Their strategies include keeping communication channels open, using consistent development environments and machine configurations, bringing offshore project leads onsite for meetings, developing and using necessary infrastructure and tools, and managing where the control and domain expertise lies. They also point out that there are some drawbacks that are difficult to overcome and should be expected such as the need for more documentation, more planning for meetings, higher levels of management overhead, and cultural differences. This work was based on an offshoring relationship with a separate vendor and not collaboration between two entities within the same company. We expect that the challenges faced in distributed development may differ based on the type of relationship between distributed sites.

Our study examines distributed development in the context of one commercial entity, which differs greatly from both open source projects and outsourcing relationships.

Issues and solutions

In his book on global software teams [4], Carmel categorizes project risk factors into four categories that act as centrifugal forces that pull global projects apart. These are:

- Loss of communication richness
- Coordination breakdowns
- Geographic dispersion
- Cultural Differences

In 2001, Battin *et al* [1] discuss the challenges and their solutions relative to each of Carmel's categories in a large scale project implementing the 3G Trial (Third Generation Cellular System) at Motorola. By addressing these challenges in this project, they found that globally distributed software development did not increase defect density, and in fact, had lower defect density than the industrial average. Table 1 lists the various locations, the size of the code developed at those locations, and their defect density. They summarize the key actions necessary for success with global development in order of importance:

- Use Liaisons
- Distribute entire things for entire lifecycle
- Plan to accommodate time and distance

Carmel and Agarwal [5] present three tactics for alleviating distance in global software development, each with examples, possible solutions, and caveats:

- Reduce intensive collaboration

Development Locations	Code Size (KLOC C/C++)	Defect Density (defects/KLOC)
Beijing, China	57	0.7
Arlington Heights, US	54	0.8
Arlington Heights, US	74	1.3
Tokyo, Japan	56	0.2
Bangalore, India	165	0.5
Singapore	45	0.1
Adelaide, Australia	60	0.5

Table 1. Locations, code size, and defect density from Motorola's 3G trial project for each site

- Reduce national and organizational cultural distance
- Reduce temporal distance

Nagappan *et al* investigated the influence of organizational structure on software quality in Windows Vista [20]. They found a strong relationship between how development is distributed across the organizational structure and number of post-release failures in binaries shipped with the operating system. Along with other organizational measures, they measured the level of code ownership by the organization that the binary owner belonged to, the number of organizations that contributed at least 10% to the binary, and the organizational level of the person whose reporting engineers perform more than 75% of the edits. Our paper complements this study by examining geographically, rather than organizationally distributed development.

4. Methods and Analysis

In this section we describe our methods of gathering data for our study and the analysis methods used to evaluate our hypotheses regarding distributed development in Windows Vista.

4.1. Data Collection

Windows Vista is a large commercial software project involving nearly 3,000 developers. It comprises over 3,300+ unique binaries (defined as individual files containing machine code such as executables or a libraries) with a source code base of over 60 MLOC. Developers were distributed across 59 buildings and 21 campuses in Asia, Europe and North America. Finally, it was developed completely in-house without any outsourced elements.

Our data is focused on 3 properties: code quality, geographical location, and code ownership. Our measure of code quality is post-release failures, since these matter most to end-users. These failures are recorded for the six months following the release of Vista at the binary level as shown in figure 1.

Geographical location information for each software developer at Microsoft is obtained from the people management software at the time of release to manufacturing of

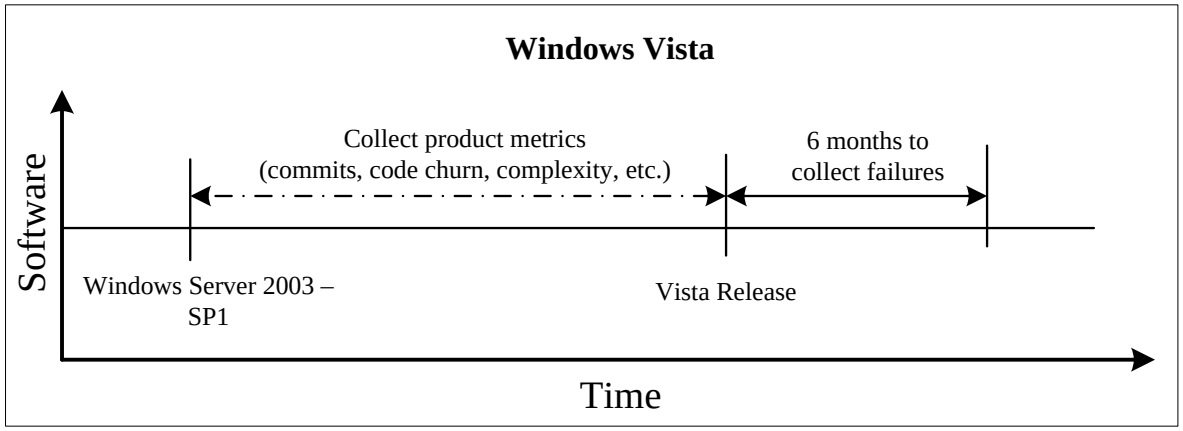

Figure 1. Data collection timeline

Vista. This data includes the building, campus, region, country, and continent information. While some developers occasionally move, it is standard practice at Microsoft to keep a software engineer at one location during an entire product cycle. Most of the 2,757 developers of Vista didn't move during the observation period.

Finally we gathered the number of commits made by each engineer to each binary. We remove build engineers from the analysis because their changes are broad, but not substantive. Many files have fields that need to be updated prior to a build, but the actual source code is not modified. By combining this data with developer geographical data, we determine the level of distribution of each binary and categorize these levels into a hierarchy. Microsoft practices a strong code ownership development process. We found that on average, 49% of the commits for a particular binary can be attributed to one engineer. Although we are basing our analysis on data from the development phase, in most cases, this is indicative of the distribution that was present during the design phase as well.

We categorized the distribution of binaries into the following geographic levels. Our reasoning behind this classification is explained below.

Building: Developers who work in the same building (and often the same floor) will enjoy more face to face and informal contact. A binary classified at the building level may have been worked on by developers on different floors of the same building.

Cafeteria: Several buildings share a cafeteria. One cafeteria services between one and five nearby buildings. Developers in different, but nearby buildings, may "share meals" together or meet by chance during meal times. In addition, the typically shorter geographical distance facilitates impromptu meetings.

Campus: A campus represents a group of buildings in one location. For instance, in the US, there are multiple campuses. Some campuses reside in the same city, but the distinction is that it is easy to travel between buildings on the same campus by foot while travel between campuses even in the same city requires a vehicle.

Locality: We use localities to represent groups of campuses that are geographically close to each other. For instance, the Seattle locality contains all of the campuses in western Washington. It's possible to travel within a locality by car on day trips, but travel between localities often requires air travel and multi-day trips. Also, all sites in a particular locality operate in the same time zone, making coordination and communication within a locality easier than between localities.

Continent: All of the locations on a given continent fall into this category. We choose to group at the continent level rather than the country level because Microsoft has offices in Vancouver Canada and we wanted those to be grouped together with other west coast sites (Seattle to Vancouver is less than 3 hours by road). If developers are located in the same continent, but not the same region, then it is likely that cultural similarities exists, but they operate in different time zones and rarely speak face to face.

World: Binaries developed by engineers on different continents are placed in this category. This level of geographical distribution means that face to face meetings are rare and synchronous communication such as phone calls or online chats are hindered by time differences. Also, cultural and language differences are more likely.

For every level of geographical dispersion there are more than two entities from the lower level within that level. That is, Vista was developed in more that three continents, localities, etc. Each binary is assigned the lowest level in the hierarchy from which at least 75% of the commits were made. Thus, if engineers residing in one region make at least 75% of the commits for a binary, but there is no campus that accounts for 75%, then the binary is categorized at the region level. This threshold was chosen based on results of prior work on development distributed across organiza-

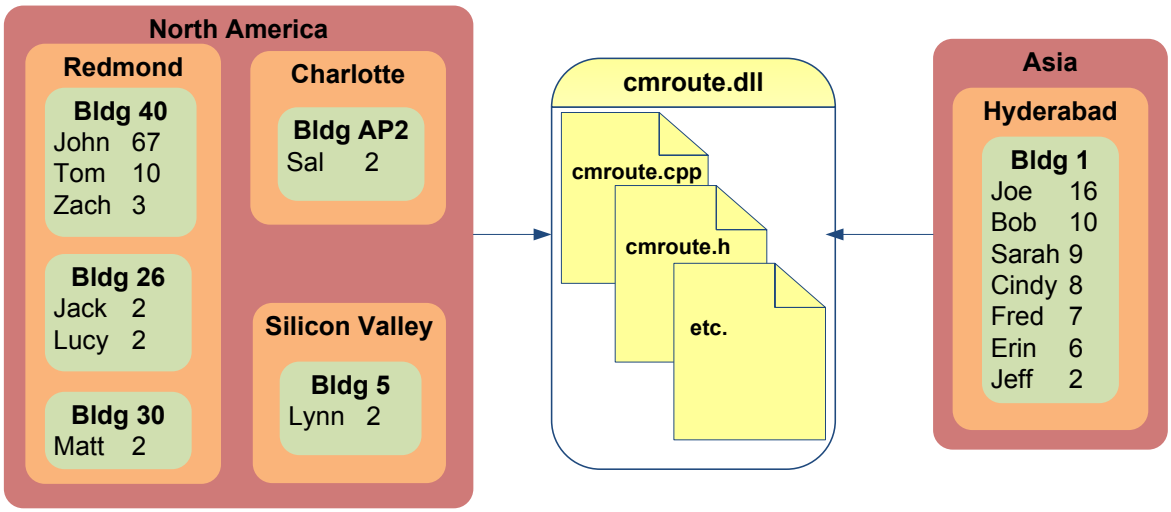

Figure 2. Commits to the library cmroute.dll. For clarity, location of anonymized developers is shown only in terms of continents, regions, and buildings.

tional boundaries that is standardized across Windows [20]. Figure 2 illustrates the geographic distribution of commits to an actual binary (with names anonymized). To assess the sensitivity of our results to this selection and address any threats to validity we performed the analysis using thresholds of 60%, 75%, 90%, and 100% with consistently similar results.

Note that whether a binary is distributed or not is orthogonal to the actual location where it was developed. For instance, some binaries that are classified at the building level were developed entirely in a building in Hyderabad, India while others were owned in Redmond, Washington.

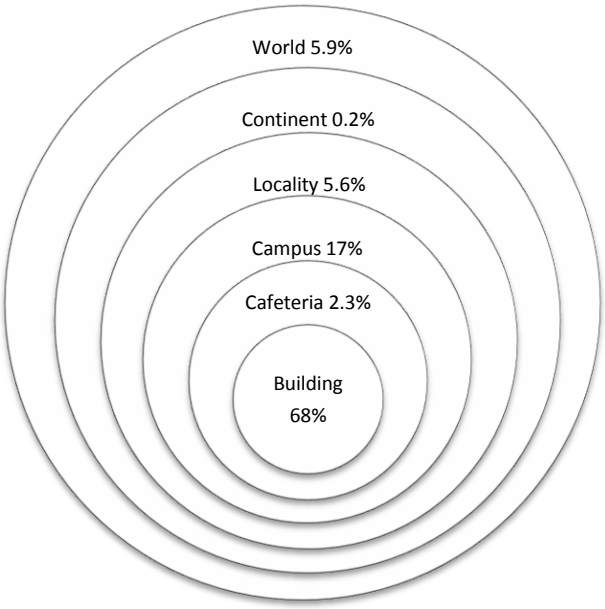

Figure 3. Hierarchy of distribution levels in Windows Vista

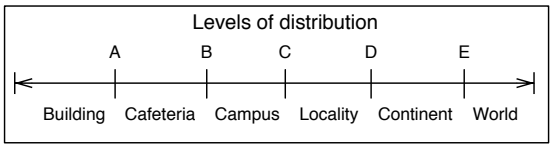

Figure 4. distribution levels in Windows Vista

Figure 3 illustrates the hierarchy and shows the proportion of binaries that fall into each category. Note that the majority of binaries have over 75% of their commits coming from just one building. The reason that so few binaries fall into the continent level is that the Unites States is the only country which contains multiple regions. Although the proportion of binaries categorized above the campus level is barely 10%, this still represents a sample of over 380 binaries; enough for a strong level of statistical power.

We initially examined the number of binaries and distribution of failures for each level of our hierarchy. In addition, we divided the binaries into ”distributed” and ”collocated” categories in five different ways using splits A through E as shown in figure 4 (e.g. split B categorizes building and cafeteria level binaries as collocated and the rest as distributed). This was performed to see if there is a level of distribution above which there is a noticeable increase in failures.

These categorizations are used to determine if there is a level of distributedness above which there is a significant increase in the number of failures. The results from analysis of these dichotomized data sets were consistent in nearly all respects. We therefore present the results of the first data set and point out deviations between the data sets where they occurred.

4.2. Experimental Analysis

In order to test our hypothesis about the difference in code quality between distributed and collocated develop-

ment, we examined the distribution of the number of post-release failures per binary in both populations. Figure 5 shows histograms of the number of bugs for distributed and non-distributed binaries. Absolute numbers are omitted from the histograms for confidentiality, but the horizontal and vertical scales are the same for both histograms. A visual inspection indicates that although the mass is different, with more binaries categorized as collocated than distributed, the distribution of failures are very similar.

A Mann-Whitney test was used to quantitatively measure the difference in means because the number of failures was not normally distributed [18]. The difference in means is statistically significant, but small. While the average number of failures per binary is higher when the binary was distributed, the actual magnitude of the increase is only about 8%. In a prior study by Herbsleb and Mockus [12], time to resolution of modification requests was positively correlated with the level of distribution of the participants. After further analysis, they discovered that the level of distribution was not significant when controlling for the number of people participating. We performed a similar analysis on our data.

We used linear regression to examine the effect of distributed development on number of failures. Our initial model contained only the binary variable indicating whether or not the binary was distributed. The number of developers working on a binary was then added to the model and we examined the coefficients in the model and amount of variance in failures explained by the predictor variables. In these models, *distributed* is a binary variable indicating if the binary is distributed and *numdevs* is the number of developers that worked on the binary. We show here the results of analysis when splitting the binaries at the regions level. The F-statistic and p value show how likely the null hypothesis (the hypothesis that the predictor variable has no effect on the response variable) is. We give the percentage increase in failures when the binaries are distributed based on the parameter values. As *numdevs* is only included in the models to examine effect of distribution when controlling for number of developers we do not include estimates or percentage increase.

In models 1 - 4 we give the percentage increase in failures when the binaries are distributed based on the parameter values also.

Model 1 F Statistic = 12.43, $p < .0005$

Variable	% increase	Std Err.	Significance
(Constant)		0.30	$p < .0005$
distributed	9.2%	0.31	$p < .0005$

This indicates that on average, a distributed binary has 9.2% more failures than a collocated binary. However, the result changes then controlling for the number of developers working on a binary.

We performed this analysis on all five splits of the binaries as shown in figure 4. The estimates for *distributed* coefficient for all models were below 17%, and dropped even further to below 9% when controlling for number of developers (many were below this value, but the numbers cited are upper bounds). In addition, the effect of *distributed* in models that accounted for the number of developers was only statistically significant when dividing binaries at the continents level.

Model 2 F Statistic = 720.74, $p < .0005$

Variable	% increase	Std Err.	Significance
(Constant)		0.25	$p < .0005$
distributed	4.6%	0.25	$p = .056$
numdevs		0.00	$p < .0005$

We also used linear regression to examine the effect of the level of distribution on the number of failures of a binary. Since the level of distribution is a nominal variable that can take on six different values, we encode it into five binary variables. The variable *diff_buildings* is 1 if the binary was distributed among different buildings that all were served by the same cafeteria and 0 otherwise. A value of 1 for *diff_cafeterias* means that the binary was developed by engineers who were served by multiple cafeterias, but were located on the same campus, etc. The percentage increase for each *diff* represents the increase in failures relative to binaries that are developed by engineers in the same building.

Model 3 F Statistic = 25.48, $p < .0005$

Variable	% increase	Std Err.	Significance
(Constant)		0.09	$p < .0005$
diff_buildings	15.1%	0.50	$p < .0005$
diff_cafeterias	16.3%	0.21	$p < .0005$
diff_campuses	12.6%	0.35	$p < .0005$
diff_localities	2.6%	1.47	$p = .824$
diff_continents	-5.1%	0.31	$p = .045$

The parameter estimates of the model indicate that binaries developed by engineers on the same campus served by different cafeterias have, on average, 16% more post-release failures than binaries developed in the same building. Interestingly, the change in number of failures is quite low for those developed in multiple regions and continents. However, when controlling for development team size, only binaries categorized at the levels of different cafeterias and different campuses show a statistically significant increase in failures over binaries developed in the same building. Even so, the actual effects are relatively minor (4% and 6% respectively).

Two important observations can be made from these models. The first is that the variance explained by the predictor variables (as measured in the adjusted R^2 value) for the built models rises from 2% and 4% (models 1 and 3) to 33% (models 2 and 4) when adding the number of developers. The second is that when controlling for the number of developers, not all levels of distribution show a sig-

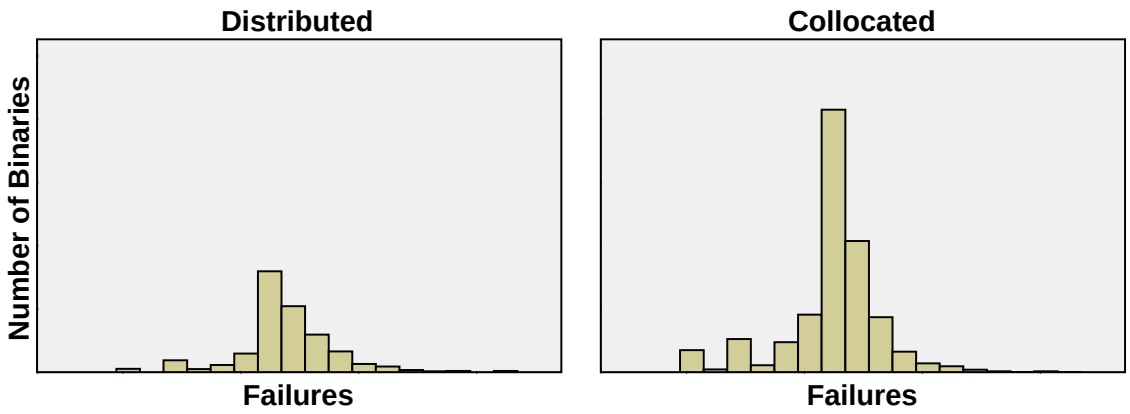

Figure 5. Histograms of the number of failures per binary for distributed (left) and collocated (right) binaries. Although numbers are not shown on the axes, the scales are the same in both histograms.

Model 4 F Statistic = 242.73, $p < .0005$			
Variable	**% increase**	**Std Err.**	**Significance**
(Constant)		0.09	$p < .0005$
diff_buildings	2.6%	0.42	$p = .493$
diff_cafeterias	3.9%	0.18	$p = .016$
diff_campuses	6.3%	0.29	$p = .019$
diff_localities	8.3%	1.23	$p = .457$
diff_continents	-3.9%	0.26	$p = .101$
numdevs		0.00	$p < .0005$

nificant effect, but the increase in post-release failures for those that do is minimal with values at or below 6%. To put this into perspective, a binary with 4 failures if collocated would have 4.24 failures if distributed. Although our response variable is different from Herbsleb and Mockus, our findings are consistent with their result that when controlling for the number of people working on a development task, distribution does not have a large effect. Based on these results, we are unable to reject the null hypothesis and H1 is not confirmed.

This leads to the surprising conclusion that in the context in which Windows Vista was developed, teams that were distributed wrote code that had virtually the same number of post-release failures as those that were collocated.

4.3. Differences in Binaries

One possible explanation for this lack of difference in failures could be that distributed binaries are smaller, less complex, have fewer dependencies, etc. Although the number of failures changes only minimally when the binaries are distributed, we are interested in the differences in characteristics between distributed and non-distributed binaries. This was done to determine if informed decisions were made about which binaries should be developed in a distributed manner. For instance, prior work has shown that the number of failures is highly correlated with code complexity and number of dependencies [19, 26]. Therefore, it is possible that in an effort to mitigate the perceived dangers of distributed development, only less complex binaries or those with less dependencies were chosen.

We also gathered metrics for each of the binaries in an attempt to determine if there is a difference in the nature of binaries that are distributed. These measures fall into 5 broad categories.

Size & Complexity: Our code size and complexity measures include number of independent paths through the code, number of functions, classes, parameters, blocks, lines, local and global variables, and cyclomatic complexity. From the call graph we extract the fan in and fan out of each function. For object oriented code we include measures of class coupling, inheritance depth, the number of base classes, subclasses and class methods, and the number of public, protected, and private data members and methods. All of these are measured as totals for the whole binary and as maximums on a function or class basis as applicable.

Code Churn: As measures of code churn we examine the change in size of the binary, the frequency of edits and the churn size in terms of lines removed, added, and modified from the beginning of Vista development to RTM.

Test Coverage: The number of blocks and arcs as well as the block coverage and arc coverage are recorded during the testing cycle for each binary.

Dependencies: Many binaries have dependencies on one another (in the form of method calls, data types, registry values that are read or written, etc.). We calculate the number of direct incoming and outgoing dependencies as well as the transitive closer of these dependencies. The depth in

the dependency graph is also recorded.

People: We include a number of statistics on the people and organizations that worked on the binaries. These include all of the metrics in our prior organizational metrics paper [20] such as the number of engineers that worked on the binary.

We began with a manual inspection of the 20 binaries with the least and 20 binaries with the most number of post-release failures in both the distributed and non-distributed categories and examined the values of the metrics described above. The only discernible differences were metrics relative to the number of people working on the code, such as team size.

Metric	Avg Value	Correlation	Significance
Functions	895.86	0.114	$p < .0005$
Complexity	4603.20	0.069	$p < .0005$
Churn Size	53430	0.057	$p = .033$
Edits	63.82	0.134	$p < .0005$
Indegree	13.04	-0.024	$p = .363$
Outdegree	9.67	0.100	$p < .0005$
Number of Devs	21.55	0.183	$p < .0005$

We evaluated the effect of these metrics on level of distribution in the entire population by examining the spearman rank correlation of distribution level of binaries (not limited to the "top 20" lists) with the code metrics. Most metrics had correlation levels below 0.1 and the few that were above that level, such as number of engineers never exceeded 0.25. Logistic regression was used to examine the relationship of the development metrics with distribution level. The increase in classification accuracy between a naive model including no independent variables and a stepwise refined model with 15 variables was only 4%. When removing data related to people that worked on the source, the refined model's accuracy only improved 2.7% from the naive model. We include the average values for a representative sample of the metrics along with a spearman rank correlation with the level of distribution for the binaries and the significance of the correlation. Although the p-values are quite low, the magnitude of the correlation is small. This is attributable to the very large sample of binaries (over 3,000).

All of these results lead to the conclusion that there is no discernible difference in the measured metrics between binaries that are distributed and those that aren't.

5. Discussion

We have presented an unexpected, but very encouraging result: it is possible to conduct in-house globalized distributed development without adversely impacting quality. It is certainly important to understand why this occurred, and how this experience can be repeated in other projects and contexts. To prime this future endeavor, in this section, we make some observations concerning pertinent practices that could have improved communication, co-ordination, team cohesion, etc., and reduced the impact of differences in culture and business context. These observations come from discussions from management as well as senior and experienced employees at many of the development sites.

Relationship Between Sites: Much of the work on distributed development examines outsourcing relationships [7, 2]. Other work has looked at strategic partnerships between companies or scenarios in which a foreign remote site was recently acquired [11]. All of these create situations where there the relationships are asymmetric and engineers at different sites may feel competitive or may for other reasons be less likely to help each other. In our situation, each of the sites has existed for a long time and has worked on software together for many years. There is no threat that if one site performs better, the other will be shut down. The pay scale and benefits to employees are equivalent at all sites in the company.

Cultural Barriers: In a study of distributed development within Lucent at sites in Great Britain and Germany, Herbsleb and Grinter [11] found that significant national cultural barriers existed. These led to a lack of trust between sites and misinterpreted actions due to lack of cultural context. This problem was alleviated when a number of engineers (liaisons) from one site visited another for an extended period of time. Battin *et al.* [1] found that when people from different sites spent time working together in close proximity, many issues such as trust, perceived ability, delayed response to communication requests, etc. were assuaged.

A similar strategy was used during the development of Vista. Development occurred mostly in the US (predominantly in Redmond) and Hyderabad, India. In the initial development phases, a number of engineers and executives left Redmond to work at the Indian site. These people had a long history, many with 10+ years within Microsoft. They understood the company's development process and had domain expertise. In addition, the majority of these employees were originally from India, removing one key challenge from globally distributed work. These people could therefore act as facilitators, information brokers, recommenders, and cultural liaisons [5] and had already garnered a high level of trust and confidence from the engineers in the US. Despite constituting only a small percent of the Indian workforce, they helped to reduce both organizational and national cultural distances [5].

Communication: Communication is the single most referenced problem in globally distributed development. Face to face meetings are difficult and rare and people are less likely to communicate with others that they don't know personally. In addition, distributed sites are more likely to use asynchronous communication channels such as email which introduce a task resolution delay [24]. A prior study of global software servicing within Microsoft has examined the ways in which distributed developers overcome communication difficulties and information needs through the use of knowledge bases and requesting help from local and

remote personnel [3].

The Vista developers made heavy use of synchronous communication daily. Employees took on the responsibility of staying at work late or arriving early for a status conference call on a rotating basis, changing the site that needed to keep odd hours every week. Keeping in close and frequent contact increases the level of awareness and the feeling of "teamness" [1, 5]. This also helps to convey status and resolve issues quickly before they escalate. In addition, Engineers also regularly traveled between remote sites during development for important meetings.

Consistent Use of Tools: Both Battin [1] and Herbsleb [12] cite the importance of the configuration management tools used. In the case of Motorola's project, a single, distributed configuration management tool was used with great success. At Lucent, each site used their own management tools, which led to an initial phase of rapid development at the cost of very cumbersome and inefficient integration work towards the end. Microsoft employs the use of one configuration management and build system throughout all of its sites. Thus every engineer is familiar with the same source code management tools, development environment, documentation method, defect tracking system, and integration process. In addition, the integration process for code is incremental, allowing problems to surface before making it into a complete build of the entire system.

End to End Ownership: One problem with distributed development is distributed ownership. When an entity fails, needs testing, or requires a modification, it may not be clear who is responsible for performing the task or assigning the work. Battin mentions ownership of a component for the entire lifecycle as one of three critical strategies when distributing development tasks. While there were a number of binaries that were committed to from different sites during the implementation phase of Vista, Microsoft practices strong code ownership. Thus, one developer is clearly "in control" of a particular piece of code from design, through implementation, and into testing and maintenance. Effort is made to minimize the number of ownership changes that occur during development.

Common Schedules: All of the development that we examined was part of one large software project. The project was not made up of distributed modules that shipped separately. Rather Vista had a fixed release date for all parties and milestones were shared across all sites. Thus all engineers had a strong interest in working together to accomplish their tasks within common time frames.

Organizational Integration: Distributed sites in Microsoft do not operate in organizational isolation. There is no top level executive at India or China that all the engineers in those locations report to. Rather, the organizational structure spans geographical locations at low levels. It is not uncommon for engineers at multiple sites may to have a common direct manager. This, in turn, causes geographically dispersed developers to be more integrated into the company and the project. The manager can act as a facilitator between engineers who may be familiar with one another and can also spot problems due to poor coordination earlier than in an organizational structure based on geography where there is little coupling between sites. Prior work has shown that organizationally distributed development dramatically affects the number of post-release defects [20]. This organizational integration across geographic boundaries reconciles the results of that work with the conclusions reached in this study. In addition, because the same process has been used in all locations of the company for some time, organizational culture is fairly consistent across geography.

6. Threats to Validity

Construct Validity

The data collection on a system the size of Windows Vista is automated. Metrics and other data were collected using production level quality tools and we have no reason to believe that there were large errors in measurement.

Internal Validity

In section 5 we listed observations about the distributed development process used at Microsoft. While we have reason to believe that these alleviate the problems associated with distributed development, a causal relationship has not been empirically shown. Further study is required to determine to what extent each of these practices actually helps. In addition, although we attempted an exhaustive search of differences in characteristics between distributed an collocated binaries, it's possible that they differ in some way not measured by our analysis in section 4.3.

External Validity

It is unclear how well our results generalize to other situations. We examine one large project and there is a dearth of literature that examines the effect of distributed development on post-release failures. We have identified similarities in Microsoft's development process with other successful distributed projects, which **may** indicate important principles and strategies to use. There are many ways in which distributed software projects may vary and the particular characteristics must be taken into account. For instance, we have no reason to expect that a study of an outsourced project would yield the same results as ours. It is also not clear how these results relate to defect fixing in the distributed world as mentioned in prior work[3].

7. Conclusion

In our study we divide binaries based on the level of geographic dispersion of their commits. We studied the post-release failures for the Windows Vista code base and concluded that distributed development has little to no effect. We posit that this negative result is a significant finding as

it refutes, at least in the context of Vista development, conventional wisdom and widely held beliefs about distributed development. When coupled with prior work, [1, 12] our results support the conclusion that there are scenarios in which distributed development can work for large software projects. Based on earlier work [20] our study shows that Organizational differences are much stronger indicators of quality than geography. An Organizational compact but geographically distributed project might be better than an geographically close organizationally distributed project. We have presented a number of observations about the development practices at Microsoft which may mitigate some of the hurdles associated with distributed development, but no causal link has been established. There is a strong similarity between these practices and those that have worked for other teams in the past [1] as well as solutions proposed in other work [11]. Directly examining the effects of these practices is an important direction for continued research in globally distributed software development.

References

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

[2] J. M. Bhat, M. Gupta, and S. N. Murthy. Overcoming requirements engineering challenges: Lessons from offshore outsourcing. *IEEE Software*, 23(6):38–44, September/October 2006.

[3] S. Bugde, N. Nagappan, S. Rajamani, and G. Ramalingam. Global software servicing: Observational experiences at microsoft. In *IEEE International Conference on Global Software Engineering*, 2008.

[4] E. Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, 1999.

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

[6] J. Cusick and A. Prasad. A practical management and engineering approach to offshore collaboration. *IEEE Software*, 23(5):20–29, September/October 2006.

[7] K. C. Desouza, Y. Awaza, and P. Baloh. Managing knowledge in global software development efforts: Issues and practices. *IEEE Software*, 23(5):30–37, Sept/Oct 2006.

[8] C. Ebert and P. D. Neve. Surviving global software development. *IEEE Software*, 18(2):62–69, 2001.

[9] D. C. Gumm. Distribution dimensions in software development projects: a taxonomy. *IEEE Software*, 23(5):45–51, 2006.

[10] J. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. *International Conference on Software Engineering*, pages 188–198, 2007.

[11] J. Herbsleb and R. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5):63–70, 1999.

[12] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–94, 2003.

[13] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *ESEC/FSE-11: Proceedings of 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 138–137, Helsinki, Finland, 2003.

[14] J. D. Herbsleb, D. J. Paulish, and M. Bass. Global software development at siemens: experience from nine projects. In *Proceedings of the 27th International Conference on Software Engineering*, pages 524–533. ACM, 2005.

[15] H. Holmstrom, E. Conchuir, P. Agerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE'06)-Volume 00*, pages 3–11, 2006.

[16] IEEE. IEEE Standard 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. 1988.

[17] R. Kommeren and P. Parviainen. Philips experiences in global distributed software development. *Empirical Software Engineering*, 12(6):647–660, 2007.

[18] H. B. Mann and W. D. R. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[19] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the International Conference on Software Engineering*, pages 452–461, 2006.

[20] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 521–530, Leipzig, Germany, 2008.

[21] T. Nguyen, T. Wolf, and D. Damian. Global software development and delay: Does distance still matter? In *Proceedings of the International Conference on Global Software Engineering*, 2008.

[22] G. M. Olson and J. S. Olson. Distance matters. *Human-Computer Interaction*, 15(2/3):139–178, 2000.

[23] L. D. Panjer, D. Damian, and M.-A. Storey. Cooperation and coordination concerns in a distributed software development project. In *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 77–80. ACM, 2008.

[24] M. Sosa, S. Eppinger, M. Pich, D. McKendrick, S. Stout, T. Manage, and F. Insead. Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry. *Engineering Management, IEEE Transactions on*, 49(1):45–58, 2002.

[25] D. Spinellis. Global software development in the freebsd project. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 73–79, Shanghai, China, 2006.

[26] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering*, pages 531–540. ACM, 2008.