Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows

Philip J. Guo* Thomas Zimmermann+ Nachiappan Nagappan+ Brendan Murphy+

* Stanford University
+ Microsoft Research

pg@cs.stanford.edu {tzimmer, nachin, bmurphy}@microsoft.com

ABSTRACT

We performed an empirical study to characterize factors that affect which bugs get fixed in Windows Vista and Windows 7, focusing on factors related to bug report edits and relationships between people involved in handling the bug. We found that bugs reported by people with better reputations were more likely to get fixed, as were bugs handled by people on the same team and working in geographical proximity. We reinforce these quantitative results with survey feedback from 358 Microsoft employees who were involved in Windows bugs. Survey respondents also mentioned additional qualitative influences on bug fixing, such as the importance of seniority and interpersonal skills of the bug reporter.

Informed by these findings, we built a statistical model to predict the probability that a new bug will be fixed (the first known one, to the best of our knowledge). We trained it on Windows Vista bugs and got a precision of 68% and recall of 64% when predicting Windows 7 bug fixes. Engineers could use such a model to prioritize bugs during triage, to estimate developer workloads, and to decide which bugs should be closed or migrated to future product versions.

Categories and Subject Descriptors:
D.2.5 [*Software Engineering*]: Testing and Debugging; D.2.7 [*Software Engineering*]: Distribution, Maintenance, and Enhancement

General Terms:
Human Factors, Management, Measurement

1. INTRODUCTION

Validation often accounts for the majority of software development costs [18]. It encompasses activities such as debugging, testing, verification, and bug tracking. In both commercial and open-source software, large projects have to keep track of hundreds of thousands of bug reports. As of August 2009, the Mozilla bug database contains over 500,000 and the Eclipse bug database over 250,000 bug reports. On average, Mozilla received 170 and Eclipse 120 new bug reports on each day from January to July 2009.

Bug triaging is the process of deciding which bugs should get fixed, a decision that typically depends on several factors [39]: How bad is the bug's impact (*severity*)? How often does this bug occur (*frequency*)? How much *effort* would be required to implement a fix? What is the *risk* of attempting to fix this bug?

Bug triaging is a pervasive process that extends far beyond developers and testers. For example, at Microsoft, employees regularly use and report bugs on versions of Microsoft software that are under active development, a practice known as "dogfooding" [15]. Thus, managers, business staff, customer service representatives, and contractors also report and edit bugs. 24,191 people in 379 office buildings in 63 countries were involved in either opening, handling, commenting on, or resolving Windows Vista bugs. That is an *order of magnitude* greater than the ∼2,000 developers who wrote code for Vista [10].

Some bugs are purposely left unfixed, for some of the following reasons: The bug occurs rarely and affects only a few users. Changes required to fix it could be large and expensive. Fixing it could introduce new bugs (code changes that fix bugs are up to twice as likely to introduce new bugs as other kinds of changes [40]). Users might be relying on existing behaviour, and thus fixing the bug could break their systems.

Ideally, developer time should be focused on bugs that will actually get fixed. However, this is not always the case. For example, in the Eclipse bug database, unfixed bugs receive almost the same amount of attention as fixed bugs, as measured by numbers of developer comments in the bug database (4.5 comments on average vs. 5.8 comments). A better understanding of what bugs are fixed could inform the design of improved triage tools and policies that allow developers to more efficiently spend their time on bugs.

For this paper, we analyzed Windows Vista and Windows 7 bug reports to characterize and predict which bugs get fixed with respect to the *involved people* (bug reporter and assignees) and their *activities* in the bug database (e.g., report edits, reassignments, and reopenings). Existing studies of bug triaging focus on assigning bugs to developers [3, 13] and detecting duplicates [22, 37, 25, 41], but to our knowledge there is no prior published academic study that characterizes which bugs get fixed. Our study contributes to the empirical body of validated research in bug fixing and informs the design of new bug tracking tools and policies.

1.1 Contributions

After surveying related work (Section 2) and describing our experimental methodology (Section 3), we devote the bulk of this paper to three main contributions:

- **Characterization of which bugs get fixed**

  In a quantitative study of Windows Vista and Windows 7 bugs, we describe how people's reputations, bug report edit activities, and geographical and organizational relationships affect the chances of a bug getting fixed (Section 4). We summarize the findings in a logistic regression model (Section 5).

ICSE '10, May 2-8 2010, Cape Town, South Africa

- **Qualitative validation of quantitative findings**

  We conducted a survey of 1,773 Microsoft employees to obtain feedback on our characterizations, of which 358 (20%) responded. These responses corroborated our quantitative findings and enriched them with qualitative insights (Section 4). They also revealed additional influences on whether a bug gets fixed, such as report quality, perceived customer and business impact, and seniority and interpersonal skills of the bug opener (Section 7).

- **Statistical model to predict which bugs get fixed**

  We developed the first model in the research literature (to the best of our knowledge) that can predict whether a bug gets fixed (Section 6). Such predictions help to prioritize bug reports and to decide which bugs to carry over to the next release of a product. To ensure that this model can generalize, we developed it and evaluated its performance on separate datasets (Windows Vista and Windows 7, respectively). We achieved a precision of 68% and a recall of 64% when predicting Windows 7 bug fixes.

We conclude by addressing threats to validity (Section 8) and providing some recommendations for tool and policy design (Section 9).

2. RELATED WORK

Several studies modeled the lifetimes of bugs, investigating properties like time-to-resolve (how long it takes a bug report to be marked as *resolved*), where the resolution can be of any outcome (e.g., FIXED, WONTFIX, DUPLICATE, WORKSFORME). Hooimeijer and Weimer [23] built a descriptive model for the lifetime of a bug report based on self-reported severity, readability, daily load, reputation, and changes over time. Panjer [36] used information known at the beginning of a bug's lifetime such as severity, component, platform, and comments to predict its time-to-resolve. Bettenburg et al. [7] observed that bug reports are fixed sooner when they contain stack traces or are easy to read. Anbalagan and Vouk [1] found that the more people are involved in a bug report, the higher its time-to-resolve. Mockus et al. [33] found that in Apache and Mozilla, bugs with higher priority are fixed faster than bugs with lower priority. Herbsleb and Mockus [20] observed that distributed work items (e.g., bug reports) take about 2.5 times as long to resolve as co-located work items. Cataldo et al. [14] found that when coordination patterns are congruent with their coordination needs, the resolution time of modification requests (similar to bug reports) was significantly reduced. In contrast to these time-to-resolve studies, we analyze and predict the *probability that a bug is successfully resolved*, i.e, its resolution outcome is marked as FIXED.

Several groups of researchers developed techniques to prioritize static analysis warnings by building models to predict which warnings are likely to get triaged and fixed: Kim and Ernst [28] using static analysis tools for open-source Java projects, Ruthruff et al. [38] for commercial Java projects, and Guo and Engler [17] for open-source C projects. In contrast to these works, we predict the likelihood of fix for bug reports in general and not just for static analysis warnings. In addition, we consider in our models bug report edit activities as well as people-related factors like reputation and geographic and organizational relationships.

Several studies characterized properties of bug reports and their edit activities: Bettenburg et al. [7] characterized what makes a good bug report. By contrast, this paper analyzes which bug reports get fixed. While the quality of a bug report certainly influences the chances of getting the bug fixed, there are additional factors discussed in this paper such as edit activities, developer relationships, and severity. Aranda and Venolia [5] examined communication between developers about bug reports at Microsoft to identify common bug fixing coordination patterns. Breu et al. [11] categorized questions asked in open-source bug reports and analyzed response rates and times by category. Bettenburg et al. [8] quantified the amount of additional information in bug duplicates. Jeong et al. [26] analyzed the reassignment of bug reports (called bug tossing) and developed tossing graphs to support bug triaging activities. Ko et al. [29] conducted a linguistic analysis of bug report titles and observed a large degree of regularity. However, none of these studies characterized and predicted which bugs get fixed.

To improve bug triaging, previous research proposed techniques to semi-automatically assign developers to bug reports [3, 13, 4], assign locations to bug reports [12], recognize bug duplicates [22, 37, 25, 41], assess the severity of bugs [32], and predict effort for bug reports [42]. With this paper, we provide support for another triaging activity: deciding which bugs should get fixed.

Independently from us, Diederik van Liere explored factors that affected which bugs were fixed in Mozilla Firefox; he published his findings on his blog [31]. In another study, he investigated how the information provided by open-source community members influences the repair time of software defects [30].

Empirical studies allow us to build a validated body of knowledge in software engineering and are crucial for informing the design of new bug tracking tools. This paper adds a characterization of what makes bug reports successful to that body of knowledge.

3. METHODOLOGY

For this study, we performed a quantitative analysis of bug datasets (Section 4), built logistic regression models using features extracted from them (Sections 5 and 6), and analyzed results from a survey consisting of both multiple-choice and free-response questions (Sections 4 and 7). Here are our data sources:

Windows Vista bug database: Our primary data source was the set of bug reports for Windows Vista, containing all pre- and post-release bugs collected in July 2009 (2.5 years after Vista's release date). We consider our dataset to be fairly complete for the factors we want to investigate, since very few new Vista bugs are being opened, compared to when it was under active development. For confidentiality reasons, we cannot reveal the exact number of reports, but it is at least an order of magnitude larger than datasets used in related work [23].

For each bug report, we extracted a list of edit events that occurred throughout its lifetime. Each event alters one or more of these fields in the bug report:

- **Editor**: Who made this edit?
- **State**: OPENED, RESOLVED, or CLOSED
- **Bug source**: How was the bug found? See Table 1 for details.
- **Bug type**: What kind of bug is it? e.g., bug in code, specification, documentation, or test suite
- **Component path**: Which component is the bug in? e.g., `Desktop Shell/Navigation/Start Menu`
- **Severity**: An indicator of the bug's potential impact on customers. Crashes, hangs, and security exploits have the highest severity (Level 4); minor UI blemishes, typos, or trivial cosmetic bugs have the lowest severity (Level 1).
- **Opener**: Who opened this bug?
- **Assignee**: Who is now assigned to handle this bug?

Bug source	Description
Human review	code, design, spec, security reviews
Code analysis tool	automated program analysis tools
Ad-hoc testing	exploratory and ad-hoc testing
Component testing	unit tests, model-based tests
System testing	integration, build, stress tests
Customer	reported by non-Microsoft user
Internal user	reported by Microsoft user

Table 1: Sources of Windows Vista bugs

- **Resolution status**: How was this bug resolved? For this study, we only care whether a bug was resolved as FIXED or not. Other statuses include DUPLICATE and WON'T FIX. (null if state is not RESOLVED)

We excluded fields that were edited infrequently or unreliably. Most notably, we excluded the *Priority* field, which represents a subjective rating of bug importance, since there was too much inconsistency in how different teams within Microsoft used it (there are similar inconsistencies in open-source projects [21]).

Here is a typical bug's life cycle: When it is first opened, all of its fields except for *Resolution status* are set. Then the bug might be edited a few times (e.g., to upgrade its severity). A special type of edit called a **reassignment** occurs when the *Assignee* field is edited. When somebody thinks he/she has resolved the bug, its *Resolution status* field is set. After the resolution attempt is approved (usually by the opener), the bug is closed. However, it might be **reopened** if the problem has not actually been properly resolved.

The main goal of our study is to characterize what factors lead a bug to be *successfully* resolved as FIXED (as opposed to related studies like Hooimeijer and Weimer [23] that investigate whether a bug is resolved in *any manner* within a set period of time). When a bug has been resolved more than once (i.e., it was reopened), we use its final resolution status to determine whether it has been successfully resolved.

Geographical data: To correlate location-related factors with bug fixes, we obtained from people management software the *building* and *country* where each employee worked when Vista was being developed. Sometimes people switch locations, but in general Microsoft tries to keep employees in the same location during a product cycle [10]. Like Bird et al. [10], we also considered *campus* (e.g., Redmond, Silicon Valley) and *continent* as factors, but our findings using those were so similar to simply using country that we omitted them for brevity.

Organizational data: Nagappan et al. [34] found that organizational factors are correlated with bugs. Motivated by that study, we obtained the name of each employee's manager, which allows us to build an organizational tree and determine whether two employees are part of the same team (i.e., share the same manager).

Microsoft employee survey: To get feedback about the fidelity of our quantitative results, we sent out an online survey to 1,773 Microsoft employees. Since we wanted to get the opinions of people well-versed in handling Windows bugs, we chose as our survey participants the top 10% of people who have opened, been assigned to, and resolved Vista bugs.

The main question in our survey was: *In your experience, how do each of these factors affect the chances of whether a bug will get successfully resolved as FIXED?* We chose the factors based on our quantitative findings, and asked respondents to rate the effect of each on bug fixes on a 7-point Likert scale, ranging from "Significantly increase chances of fix" to "Significantly decrease chances of fix" (we also included a "No opinion" option).

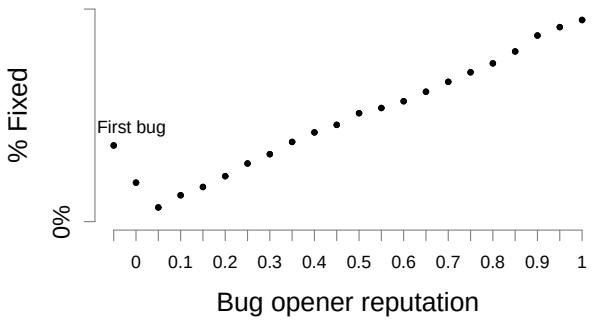

Figure 1: Percent of fixed Vista bugs vs. bug opener reputation (rounded up to nearest 0.05). "First bug" represents all bugs whose opener had never opened a bug before the current one. The y-axis is unlabeled for confidentiality.

We concluded the survey with 3 free-response questions: What are reasons why a bug might be reassigned multiple times, why a bug might be closed then reopened multiple times, and any other comments about what factors influence whether a bug gets fixed.

We received 358 responses (20% response rate). Most respondents were either developers (55%) or testers (30%). Most were fairly experienced, with a median of 11.5 years of work experience in the software industry and 9 years at Microsoft. We integrate our survey results into the findings of Section 4 and focus on the free-response answers in Section 7.

Replicated study on Windows 7: After completing our study on Windows Vista, we extracted bug reports from its successor project, Windows 7, and replicated our study on it. We used the entire bug database for the development period of Windows 7 ($\sim$3 years).

4. INFLUENCES ON BUG-FIX LIKELIHOOD

In this section, we present factors related to people and bug report edit activity that affect the likelihood of a bug being fixed. Our findings are consistent across both Windows Vista and Windows 7, but here we only present the Vista results, for space reasons (Section 4.6 summarizes some differences with Windows 7).

4.1 Reputations of bug opener and 1st assignee

We found that a bug opened by someone who has been successful in getting his/her bugs fixed in the past (has a better reputation with respect to bug reporting) is more likely to be fixed. We quantify reputation using the same metric as Hooimeijer and Weimer [23][1]:

$$\text{bug opener reputation} = \frac{|\text{OPENED} \bigcap \text{FIXED}|}{|\text{OPENED}| + 1}$$

For each bug, we calculate its opener's reputation by dividing the number of *previous bugs* that he/she has opened and gotten fixed by the total number of previous bugs he/she has opened (+1). Adding 1 to the denominator prevents divide-by-zero (for people who have never previously opened any bugs) and, more importantly, prevents

[1]We copied their reputation metric verbatim, to facilitate comparisons with related work. Building a robust, meaningful, and fair reputation metric is difficult and beyond the scope of this project.

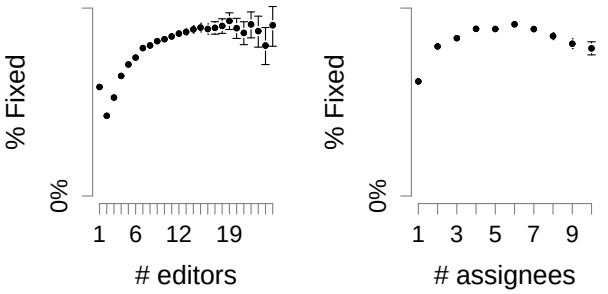

Figure 2: Percent of fixed Vista bugs vs. numbers of unique bug report editors and assignees. Vertical bars are 95% confidence intervals for binomial probabilities (most are invisible). The y-axes are unlabeled for confidentiality.

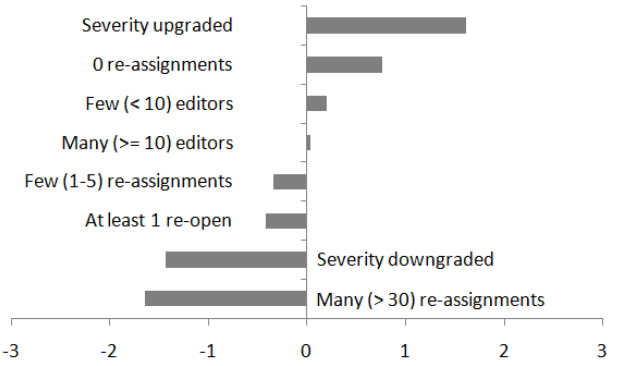

Figure 3: Likert scores for survey questions about the perceived impact of bug report edits on bug fixes, averaged over all 358 respondents. -3 means "Significantly decrease chances of fix" and $+3$ means "Significantly increase chances of fix".

people who have opened very few bugs from earning high reputations: Without the extra $+1$ term, someone who has opened 1 bug and gotten it fixed will have the same reputation as someone who has opened 100 bugs and gotten all 100 fixed; intuitively, the latter person should have a higher reputation (which our metric ensures).

In Figure 1, we grouped Windows Vista bugs by ranges of opener reputations and plotted the percentage of bugs in each group that were fixed. The leftmost two points were specially calculated: The "First bug" point considers all bugs where the opener had not opened any bugs before this one. The "0" point considers all bugs where the opener had opened some bugs but had gotten none fixed. The rest of the points consider all bugs with opener reputations within a 0.05-point range. For instance, the rightmost point represents all bugs with opener reputations between 0.95 and 1.

There is a consistent monotonic increase in bug-fix likelihood as the opener reputation increases, with the exception of those with zero reputation. Interestingly, bugs opened by first-timers ("First bug" point) had a higher likelihood of fix than bugs opened by people with low reputations. All differences between points are statistically significant since their 95% confidence intervals for binomial probabilities [16] never overlap. In fact, all confidence intervals are negligible, so they are invisible in Figure 1.

Employee intuitions corroborated our data. For the final free-response survey question on what factors influence bug fixes, one respondent (who was not aware of our quantitative findings) wrote:

> *"A big influence [on bug fixes] is the "reputation" of the person opening the bug. I often see nonsense bugs get opened indicating opener had done zero work. Contrast this with some very high quality bugs I see from some folk who have done a bunch of background work and often target the exact underlying problem. If submitter has a history of submitting high quality bugs then new bugs from that person get better attention [...]"*

When we repeated these calculations for the reputations of the first person who was assigned to each bug, the trends were nearly identical. This result shows that certain people are more effective at either fixing bugs or reassigning bugs to others who can fix them.

***Summary**: People who have been more successful in getting their bugs fixed in the past (perhaps because they wrote better bug reports) will be more likely to get their bugs fixed in the future.*

4.2 Bug report edits and editors

Each act of editing a bug report field (especially by a new editor) provides evidence that somebody is taking an interest in it and improves its likelihood of being fixed. It doesn't take much effort to edit a bug report field (e.g., to upgrade its severity), since an editor isn't necessarily the person assigned to handle the bug. Bugs with at least 1 edit are 50% more likely to be fixed as those with no edits (all two-group ratio differences reported in this paper are statistically significant at $p < 0.001$ according to a chi-square test [16]). However, too many edits might indicate a dispute over how to handle the bug, so it might not be fixed: Bugs with over 50 edits are 24% less likely to be fixed than those with fewer than 50 edits.

There is a similar positive correlation between the number of unique people who have edited a bug report and its likelihood of being fixed. The left half of Figure 2 shows that if a bug report has more editors (up to around 15), its likelihood of being fixed steadily increases. The differences are all statistically significant since the confidence intervals never overlap (most are invisible). The percent of fixed bugs seems to flatten out at around 15 editors, but the sample sizes grow too small to provide statistical power.

Contrary to our data, our survey respondents thought that the number of editors had only a weak effect on the likelihood that a bug would be fixed (Figure 3). This perception could exist because developers aren't usually aware of how many people have edited their bug reports. However, one specific type of edit — a change in severity level — elicited stronger responses. Respondents thought that a severity upgrade greatly increased chances of fix (mean Likert score of 1.6), which our data corroborates: Vista bugs whose severity had been upgraded were 20% more likely to get fixed. Respondents also thought that a severity downgrade would decrease chances of fix (mean Likert score of -1.4), but our data actually showed almost no difference from the control population of bugs whose severity level never changed.

***Summary**: The more people who take an interest in a bug report, the more likely it is to be fixed.*

4.3 Bug report reassignments and assignees

A reassignment is a major type of bug report edit, since it shifts responsibility to a new person. The right half of Figure 2 shows that reassigning a bug report to up to 6 people increases its likelihood of being fixed, but the likelihood decreases with more assignees. Most confidence intervals are invisible, but as the number of assignees approaches 10, they become visible. Calculations for number of reassignments (rather than unique assignees) show similar trends: Bugs that are reassigned at least once are 18% more likely to be fixed than those that are never reassigned, but bugs that

are assigned more than 30 times are 14% less likely to be fixed than those assigned fewer than 30 times.

One explanation for this phenomenon is that people often need to reassign a bug several times (possibly across teams or locations) to find the *optimal* developer who can best address it. A survey respondent cited the following reason for reassignments:

> *"Bugs many times are exposed in the UI [user interface], but are not caused by the team writing the UI code. These bugs can pass down several layers of components before landing on a lower level component owner."*

However, too many reassignments means that no one person or team is taking responsibility for handling the bug, so it might never get fixed [26].

According to Figure 3, survey respondents concurred that having too many reassignments (> 30) is detrimental to the chances of a bug being fixed. However, they also thought that even having a few reassignments (1–5) would be detrimental, which is contradictory with our data. This perception could exist because developers associate reassignments with inefficiencies in the bug triaging process (as indicated by some other free-response answers).

***Summary**: Reassignments are not always detrimental to bug-fix likelihood; several might be needed to find the optimal bug fixer.*

4.4 Bug report reopenings

Sometimes a bug is reopened after someone has already resolved and closed it. Our data shows that bugs that are reopened up to 4 times are not any less likely to eventually be fixed. However, if a bug is reopened too many times, then it probably will not be fixed: Bugs with more than 4 reopenings are 34% less likely to be fixed than those with fewer than 4, and those with more than 5 are 46% less likely to be fixed (too few bugs had more than 6 reopenings for the differences to be statistically significant).

Consistent with our data, survey respondents felt that reopenings only had a slightly negative effect on bug-fix likelihood (Figure 3). In a free-response question on why people felt that certain bugs might need to be reopened several times before being successfully fixed, respondents cited reasons like the following: The bugs might be difficult to reproduce, so they were first closed as NOT REPRO and later reopened when someone improved report quality. The initial fix might trigger regression test failures, so the bug must be reopened to attempt another fix.

***Summary**: Reopenings are not always detrimental to bug-fix likelihood; bugs reopened up to 4 times are just as likely to get fixed.*

4.5 Organizational and geographical distance

We found that bugs assigned to someone in a different team or geographical location as the bug opener were less likely to be fixed. We quantified these effects by partitioning bugs into groups based on organizational and geographical profiles of their openers and assignees. Then we calculated the percent of bugs in each group that were successfully fixed. In Table 2, we report ratios relative to the anonymized baseline percentages X, Y, and Z. For instance, bugs opened by and initially assigned to people with different managers are 0.85 times as likely to be fixed as those opened and initially assigned to the same person (shown in bold near the top of Table 2).

At one extreme, bugs opened by and assigned to the same person are the most likely to get fixed (the X and Y baselines), since the opener is probably a developer who wants to and is able to fix his/her own bug. Bugs assigned to someone in the same team or building are almost as likely to get fixed (0.97 and 0.94 times, respectively). These colleagues can easily talk face-to-face to resolve ambiguities in bug reports and to hold each other accountable.

ORGANIZATIONAL FACTORS

Opened by and *initially* assigned to ...	
... the same person	X
... someone with the same manager	$0.97 \cdot X$
... someone with a different manager	**0.85** $\cdot X$
Assigned to opener at some point in time	Y
Never assigned to opener, but assigned to someone with the same manager as opener	$0.56 \cdot Y$
Never assigned to anyone with same manager	$0.62 \cdot Y$
Opened by a permanent employee	Z
Opened by a temporary employee	$0.84 \cdot Z$
Initially assigned to temp. employee	$0.94 \cdot Z$
Assigned to temp. employee at any point	$0.97 \cdot Z$

GEOGRAPHICAL FACTORS

Opened by and *initially* assigned to ...	
... the same person	X
... someone in the same building	$0.94 \cdot X$
... someone in a different building but in the same country	$0.77 \cdot X$
... someone in a different country	$0.79 \cdot X$
Assigned to opener at some point in time	Y
Never assigned to opener, but assigned to someone in the same building	$0.66 \cdot Y$
Never assigned to anyone in same building, but assigned to someone in the same country	$0.63 \cdot Y$
Never assigned to anyone in the same country	**0.33** $\cdot Y$

Table 2: Effect of organizational and geographical factors on percentage of Vista bugs that get fixed. Exact percentages for each group are anonymized; only ratios are shown.

However, if bugs are assigned to people who work in different buildings or countries, then there is greater overhead in communication. In a free-response question about what factors affect bug fixes, one survey respondent cites *"poor communication, language barrier problems with other countries"* as hindrances. Bugs that were never assigned to anyone in the same country as their opener were only 0.33 times as likely to be fixed as those assigned to the opener at some point in time (shown in bold at the bottom of Table 2). Survey results in Figure 4 corroborate these findings. Herbsleb and Grinter found that cultural and language barriers existed at Lucent software development sites just within Western Europe [19]; Microsoft employees in 63 countries across 6 continents were involved in Windows Vista bugs, so the potential barriers are even greater.

Also, Microsoft tries to organize teams so that all members are located in the same building. Thus, when bugs are assigned across buildings, chances are that the participants do not personally know one another. Another survey respondent pointed out that there is less implicit trust between people in different teams or locations:

> *"Personal relations between the bug opener and members of the team it is assigned to [affects bug triaging]. Whenever I open bugs assigned to people I know, they are investigated thoroughly as there is a trust in the report I write. Often when reporting a*

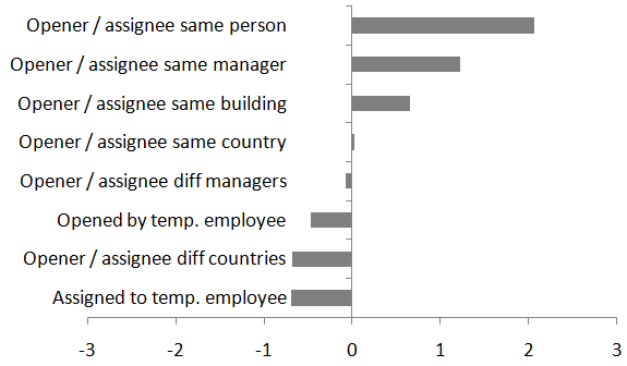

Figure 4: Likert scores for survey questions about the perceived impact of people factors on bug fixes, averaged over all 358 respondents. -3 means "Significantly decrease chances of fix" and $+3$ means "Significantly increase chances of fix".

bug within areas where I don't know the owners, there is inherent distrust in the bug report."

At another extreme, temporary employees (e.g., contractors or interns) have lower reputation and fewer relationships with Windows developers, so their bug reports might not be taken as seriously as those from core employees. Figure 4 shows that survey respondents felt that bugs opened by and assigned to temporary employees were less likely to get fixed. The section of Table 2 with the Z baseline corroborate these hunches. It's more detrimental for a bug to be opened by a temp than to be assigned to a temp, though, since the burden is on the opener to find someone to fix the bug.

As a bug is reassigned across several buildings, its likelihood of being fixed drops monotonically. There is a 7% drop in bug-fix likelihood when going from 1 to 2 buildings, and another 7% drop when going from 2 to 3 (the drops get smaller as the number of buildings increases). Recall from Figure 2 that more assignees (up to 6) actually improves likelihood of fix, but that is not true for assignee buildings.

Herbsleb and Mockus found that Modification Requests (MRs), which include bug reports, took longer to resolve when development was globally distributed [20]. Our findings supplement theirs by showing that bug reports assigned across teams, buildings, and countries are less likely to be *successfully* resolved. Herbsleb and Mockus attributed these delays in part to the fact that more people were usually involved in MRs distributed across sites. In our data, bug reports distributed across sites did not involve significantly more people (usually only one extra editor and assignee), but we still observed noticeable drops in bug-fix likelihood.

***Summary**: Bugs assigned across teams or locations are less likely to get fixed, due to less communication and lowered trust.*

4.6 Replicated study on Windows 7

Although the exact numbers and graphs for Windows 7 data are slightly different than those for Vista, the overall trends are the same. Here are some notable differences: Figure 5 shows the same plots on Windows 7 data as Figure 2 does for Vista. There is a sharper upward progression and sudden flattening in percent fixed at 11 editors and a larger spike at 4 assignees. Also, there is a much sharper drop-off in percent of fixed bugs as the number of assignee buildings increases (not pictured for space). For example, Windows 7 bugs whose assignees were distributed across 2 buildings

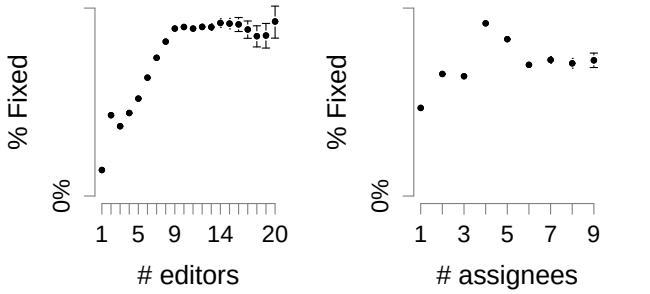

Figure 5: Percent of fixed Windows 7 bugs vs. numbers of unique bug report editors and assignees (analogous to Figure 2 for Vista bugs). Vertical bars are 95% confidence intervals for binomial probabilities (most are invisible).

were 29% less likely to get fixed than those that remained within 1 building, versus only a 7% decrease for Vista.

5. DESCRIPTIVE STATISTICAL MODEL

A problem that often arises when presenting a series of single-variable correlations (as we've done in Section 4 with factors that correlate with whether a bug is fixed) is that their effects might be cross-correlated, thereby diminishing their validity. To show that the factors we picked have *independent effects* that do not confound, we built a logistic regression model and ran an Analysis of Deviance test on all factors [24].

5.1 Model building

A logistic regression model aims to predict the probability of an event occurring (e.g., will this bug be fixed?) using a combination of factors that can be numerical (e.g., bug opener reputation), boolean (e.g., was severity upgraded?), or categorical (e.g., bug source). We built a model to predict the probability that a Windows Vista bug will be fixed, using our entire bug dataset as training data (Table 3). We chose our factors (explanatory variables) from those described in Section 4. We also built a model for Windows 7 using the same factors; we obtained almost identical results, so we do not show them here due to space constraints.

We determined that all factors had independent effects by incrementally adding each one to an empty model and observing that the model's deviance (error) decreases by a statistically significant amount for all added factors (a standard technique called *Analysis of Deviance* [24]). We found that all factors were statistically significant at $p \ll 0.001$ according to an Analysis of Deviance chi-square test.

We also checked for interactions between the factors in Table 3. In particular, we were worried about interactions between opener and assignee having the same manager and being in the same building, since people on the same team often work in the same building. We found no interactions between any pairs of factors, though.

The purpose of this model is to *describe* the various independent effects on bug fixes. Note that this model cannot actually be used to *predict* the probability that a newly-opened bug report will be fixed, since it uses factors that are not available at bug opening time (e.g., number of reopens).

5.2 Meaning of logistic regression coefficients

One main benefit of using logistic regression over other types of statistical models (e.g., support vector machines) is that its parameters (e.g., the coefficients in Table 3) have intuitive meanings.

FACTOR	COEFFICIENT
Bug source (7 categories)	
Human review	0.51
Code analysis tool	0.36
Component testing	0.07
Ad-hoc testing	†
System testing	−0.13
Customer ⋆	−0.35
Internal user ⋆	−0.45
⋆ (Relatively fewer bugs from sources marked as ⋆ were fixed, due to numerous duplicate bug reports and the difficulty of reproducing bugs reported by users in the field.)	
Reputation of bug opener	2.19
Reputation of 1st assignee	2.46
Opened by temporary employee?	−0.12
Initial severity level	0.03
Opener / any assignee same manager?	0.68
Opener / any assignee same building?	0.27
Severity upgraded?	0.26
Num. editors	0.24
Num. assignee buildings	−0.26
Num. re-opens	−0.13
Num. component path changes	−0.23

Table 3: Descriptive logistic regression model for bug-fix probability, trained on all Vista bugs. The factor labeled † folds into the intercept term, which is omitted for confidentiality.

Numerical and boolean factors: The sign of each coefficient is its *direction* of correlation with the probability of a bug being fixed. For example, bug opener reputation is positively correlated with bug fixes, so its coefficient is positive (2.19). The magnitude of each coefficient roughly indicates how much a particular factor affects bug-fix probability. See Hosmer and Lemeshow [24] for details on how to transform these coefficients into exact probabilities.

In general, it's hard to compare coefficient magnitudes across factors, since their units of measurement might differ. However, it's possible to compare coefficients for, say, two boolean factors like "Opener / any assignee same manager" and "Opener / any assignee same building". The coefficient of the former (0.68) is larger than that of the latter (0.27), which means that having the same manager has a *larger positive effect* on bug-fix probability than working in the same building.

Categorical factors: If a factor has N categories (levels), then $N-1$ of them get their own coefficient, and the remaining one gets its coefficient folded into the intercept term (the R statistics package chooses the alphabetically earliest category to fold, so that's why "Ad-hoc testing" has no coefficient in Table 3). What matters isn't the value of each coefficient but rather their *ordering* across categories. For example, for the categorical factor "Bug source", the coefficient for "Human review" is higher than that for "Code analysis tool", which means that bugs in the former category are *more likely* to be fixed than bugs in the latter.

5.3 Interpreting our descriptive model

The following factors already discussed in Section 4 are *positively correlated* with bug-fix probability, so their corresponding coefficients are positive: reputations of the bug opener and 1st assignee, whether the opener and any assignee had the same manager or worked in the same building, whether severity was upgraded, and number of bug report editors. These factors are *negatively correlated* with bug-fix probability, so their coefficients are negative: whether the bug was opened by a temporary employee and numbers of reopens and assignee buildings.

We also included in our model three additional factors that we did not have space to discuss in depth in Section 4:

Bug source: Bugs from different sources vary in how often they are fixed. We have sorted the bug source categories in Table 3 in decreasing order of coefficient values, which ranks them by how many percent of bugs in each category were fixed. In general, bugs that are easier to triage, reproduce, and confirm are fixed more frequently. On the high end, human review and code analysis bugs are easy to triage and require no separate reproduction steps. On the low end, bugs reported by users can be difficult to reproduce (users are not trained to write methodical bug reports like testers are) and are often resolved as DUPLICATE and not as FIXED (many users might encounter different symptoms of the same underlying bug).

Initial severity level: Bugs opened with a higher severity value (in the range of 1–4) are more likely to be fixed, as reflected by its 0.03 coefficient. Our Windows Vista data shows that a Severity 4 bug is 11% more likely to be fixed than a Severity 2 or 3 bug and 17% more likely to be fixed than a Severity 1 bug.

Num. component path changes: Bugs with more component path changes are less likely to be fixed, as reflected by its −0.23 coefficient. If people edit a bug report to change its component path, then that's a sign that they are unsure of the bug's root cause and/or where a fix should be applied. Our Windows Vista data shows that a bug with its path changed at least once is 12% less likely to be fixed than one with no path changes.

5.4 Excluded factors

We excluded several factors in Section 4 from our model, because they had similar yet weaker effects than factors we already included. Trying to add these factors did not further decrease our model's deviance by statistically significant amounts, so we excluded them in order to form the simplest possible model.

We used num. assignee buildings rather than num. assignees because the former had a monotonic effect while the latter did not. Similarly, the effect of num. reassigns was not monotonic. We used num. editors rather than num. edits since the former had a stronger and more consistent effect. We used buildings rather than countries since the former had a stronger effect.

6. PREDICTIVE STATISTICAL MODEL

Engineers cannot directly use the descriptive model of Section 5 to predict the probability that a newly-opened bug will be fixed, since it contains factors that are not available at open-time (e.g., num. reassignments). To remedy this limitation, we created a simple predictive model and evaluated its performance.

6.1 Model building

We built a predictive model using only factors that are available at the time a bug report is opened (Table 4). To do so, we modified our descriptive model of Table 3 by removing the last 5 factors, changing "Opener / any assignee ..." to "Opener / 1st assignee ..."[2], and re-training on our entire Windows Vista dataset to obtain

[2]At the time a bug is opened, it must be assigned to someone.

FACTOR	COEFFICIENT
Bug source (7 categories)	
Human review	0.39
Code analysis tool	0.10
Component testing	~ 0.00
Ad-hoc testing	†
System testing	-0.21
Customer ⋆	-0.35
Internal user ⋆	-0.47
⋆ (Relatively fewer bugs from sources marked as ⋆ were fixed, due to numerous duplicate bug reports and the difficulty of reproducing bugs reported by users in the field.)	
Reputation of bug opener	2.19
Reputation of 1st assignee	2.39
Opened by temporary employee?	-0.04
Initial severity level	0.06
Opener / 1st assignee same manager?	0.27
Opener / 1st assignee same building?	0.03

Table 4: Predictive logistic regression model for bug-fix probability, trained on all Vista bugs. The factor labeled † folds into the intercept term, which is omitted for confidentiality.

new coefficients. The two models are quite similar: Although the exact values of the coefficients are different, their signs and magnitudes relative to one another remain unchanged.

6.2 Model performance

We evaluated the predictive power of our model using cross-validation [35] and by predicting bug fixes on an entirely new dataset.

For cross-validation, we randomly picked 2/3 of our data to train the model, and the other 1/3 to test its accuracy. We predict a bug as being fixed only when the model gives a probability exceeding 0.5. We performed 20 rounds of randomized cross-validation and obtained an average precision of 0.67 and recall of 0.68. We also did 10-fold cross-validation [35] and found identical results.

We also ran cross-validation on our descriptive model (Table 3), and its precision and recall were only 0.04 greater than that achieved by our predictive model. This provides encouragement that factors available at the time a bug is first opened are almost as effective as factors from the entire bug lifetime.

The ideal way to evaluate model performance is to train on one dataset and test on an entirely new dataset that we didn't have access to when we were creating the model (cross-validation is an approximation to this ideal when only one dataset is available). After developing and running our analyses on the Windows Vista dataset, we extracted bug reports from Windows 7, the successor of Vista. We then trained our model on the entire Windows Vista dataset and used it to predict which bugs will be fixed in Windows 7. We obtained a precision of 0.68 and recall of 0.64. These values are comparable to those obtained with cross-validation on the Vista dataset alone, so we are confident that this model has predictive powers that generalize beyond the dataset on which it was developed.

For the proof-of-concept model presented here, we didn't focus on optimizing performance — we chose logistic regression because its parameters are relatively easy to interpret. Using more sophisticated classifiers (e.g., support vector machines) or ensemble learning methods (e.g., boosting) could improve precision and recall.

Are these precision and recall values "good enough"? Unfortunately, we cannot directly compare to related work, since we know of no prior work that provides such performance numbers for models that predict bug-fix probability. We feel that our model performs well enough to consider deploying it to help triage bugs within Microsoft. Only by collecting user feedback and understanding the role which a predictive model plays in a specific triage process can we figure out what performance numbers are adequate and how to best improve our model.

6.3 Applications of predictive model

In practice, such a model can improve bug triaging as follows:

- **Prioritizing bug reports during triage**: During software development, resources are often spent on bugs that are never fixed. Our prediction model helps to prioritize bug reports and to focus resources on the bugs that are most likely to be fixed. Thus, wasted effort can be reduced.
- **Deciding which bugs to prematurely close**: In the Mozilla project, bug reports are automatically closed after an inactivity period of 90 days [11]. Even though the auto-close feature is not very popular among users [27], it is necessary to control the number of open bugs. Our prediction model could improve this situation by closing bugs more selectively, preferring bugs that are predicted not to be fixed.

Knowing which bugs will get fixed also helps to monitor and track the progress of software projects. For example, our prediction model can be used to provide an estimate of how many of the open bugs have yet to be fixed. Similarly, the model helps to estimate the number of fixed bugs by developers; rather than just counting assigned bug reports, we can additionally estimate how many of each developer's bug reports will be fixed. Such estimates can improve automatic triaging techniques such as the one introduced by Anvik and Murphy ("Who should fix this bug?") [4], because in practice every developer can fix only a limited number of bugs.

7. ADDITIONAL INSIGHTS FROM SURVEY

Our final survey question was free-response: *Please write any other comments about what factors influence whether a bug gets successfully fixed.* 189 out of the 358 respondents answered this question. When we examined their responses, we found that many of them mentioned the following qualitative factors:

Textual quality of bug report: Precise, well-written reports are more likely to gain the triager's attention, especially if there are clear steps for reproducing the bug (confirming the findings of Bettenburg et al. [7]):

> *"Just to re-emphasize: The quality of the bug description is very important. Not necessarily filling in the dozens of fields in the bug database with all sorts of crap (build numbers, dates, classifications, etc) - but just the plain-text description of the problem, the implication and maybe even the potential solution."*

Perceived customer/business impact: Bugs that are likely to hurt the company financially or in terms of reputation amongst customers will get more attention:

> *"Customer impact can be a very big impact on a bug if evidence exists to show the cost of not fixing the bug for one or more customers."*

> *"The biggest factor is the impact on end users. If the impact on end users is high, the bug will always be fixed."*

Seniority of bug opener: Bugs reported by and affecting higher-ranked employees often get preferential treatment:

"A bug opened because something went wrong on a VPs [vice president's] laptop has better chance [of being fixed] than a bug opened because the same thing happened to an intern."

Interpersonal skills of bug opener: With limited resources available to fix bugs, people who are more persuasive champions for their bugs are more successful in getting them addressed and fixed:

"One other 'soft' factor is the speaking skill persuasiveness of the developer (or other representative) when arguing for the bug."

"How hard someone fights for the bug to be fixed (may be the opener, customer or feature owner). Whether there is someone on the team particularly passionate about the issue (or somebody very senior or influential in the company)."

8. THREATS TO VALIDITY

Construct validity: Antoniol et al. [2] pointed out that not all bug reports are related to software problems; in some case bug reports correspond to feature requests. For our study, we used the *bug type* field to distinguish between different kinds of bug reports such as code bug, spec bug, test bug, and feature request. However, since this factor had little influence on the accuracy of our models, we excluded it from the discussion in this paper.

Internal validity: In a qualitative study of 10 bugs, Aranda and Venolia [5] found that many details are discussed even before a bug report is created and that not all information is recorded in bug tracking systems. For our study, this is only a minor threat because most of the events we analyzed *must be recorded* in the bug database (e.g., reassignments and reopenings). Moreover, people make triage decisions based on data available in the bug database. We also validated our quantitative results with qualitative feedback from Microsoft employees involved in handling these bugs.

Bird et al. [9] raised the issue of bias in bug datasets for defect prediction in open-source projects. However, the likelihood of bias in our dataset is low since we analyzed the entire population of Windows Vista and Windows 7 bug reports.

External validity: Drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables [6]. For this reason, we cannot assume a priori that the results of our study generalize beyond the specific environment in which it was conducted. However, we feel that our results can potentially generalize (at least to other large-scale systems software projects) because they were consistent when we replicated the initial Windows Vista study on Windows 7.

Replication studies on projects outside of Microsoft are essential for strengthening external validity. Concurrently with our work, Diederik van Liere applied logistic regression to predict which bugs were fixed in the Mozilla Firefox project, using some factors similar to those in our models (e.g., bug opener reputation) [31]. Although his findings were only published as a short blog post, they still help improve the generality of our own findings and motivate us to replicate our study in more detail on open-source projects.

Common misinterpretation: Lastly, a common misinterpretation of empirical studies is that nothing new is learned (e.g., *"I already knew this result"*). However, such wisdom has rarely been shown to be true and is often quoted without scientific evidence. This paper provides such evidence: Most common wisdom is confirmed (e.g., *"distributed bug reports are less likely to be fixed"*) while others are challenged (e.g., *"bug reassignments are always bad"*).

9. CONCLUSIONS

Fixing bugs is a crucial software maintenance activity. It is very important for software developers and managers to know how and what types of bugs get fixed. Our study investigated factors related to people and report edit activities that affect the bug triage process.

Quantitatively, our findings characterized which bug reports get successfully fixed. To highlight, we found that at least one reassignment increases but too many reassignments decrease the likelihood of a bug getting fixed. We also observed that the higher reputation a bug opener has, the more likely his/her bugs are to get fixed, and that bugs handled by multiple teams and across multiple locations are less likely to get fixed.

Qualitatively, our survey results provide insights into difficult-to-measure social factors that affect the bug triaging process. For example, survey respondents pointed out the influence of seniority, reputation, personal relations, and trust.

Informing tool and policy design: Based on our findings, we can make the following recommendations to improve bug triaging tools and processes:

- Use prediction models to rank and filter bugs during triage.
- Improve collective awareness of each developer's areas of expertise, to minimize the number of reassignments required to find the optimal person to fix a particular bug.
- Minimize the number of times a bug must be reopened before being fixed, perhaps by improving regression testing.
- Train employees to write high-quality bug reports (highlighting the importance of reputation), which can reduce both unnecessary reassignments and reopenings.
- Reorganize teams to reduce the number of cross-team and cross-building reassignments.
- Improve communication and trust amongst people working in different teams and locations, to improve the likelihood of distributed bugs being fixed.
- Encourage more fairness and objectivity in prioritizing bugs, so that seniority and personal connections are less influential.

Future work: Looking forward, we plan to replicate this study on other projects within Microsoft and also on open-source projects, taking into account the ways that open-source project developer dynamics differ from those in commercial projects. More broadly, we plan to build a social network of developers in Windows based on their personal interactions and also a developer network of people who worked together to write code and fix bugs; using these two networks, we can assess the importance of social connections in software development and bug fixing. In the end, people are the core of any software development process, so it is crucial to understand how they work in aggregate (using quantitative methods like data analysis) as well as their perceptions of inefficiencies in their workflow (using qualitative methods like surveys). This study is one step in that direction; we hope it can inform future work on people-related factors that affect bug triaging.

Acknowledgments: Thanks to Christian Bird and ICSE reviewers for their insightful critiques. Thanks to the Microsoft Windows team for their help in understanding the data. Philip Guo performed this work during a summer internship at Microsoft Research.

10. REFERENCES

[1] P. Anbalagan and M. Vouk. On predicting the time taken to correct bugs in open source projects (short paper). In *ICSM '09: Proceedings of the 25th IEEE International Conference on Software Maintenance*, September 2009.

[2] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON '08: Proceedings of the 2008 conference of the center for advanced studies on collaborative research*, pages 304–318, 2008.

[3] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 361–370, 2006.

[4] J. Anvik and G. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*.

[5] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *ICSE' 09: Proceedings of the 31st International Conference on Software Engineering*, 2009.

[6] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Trans. Softw. Eng.*, 25(4), 1999.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *FSE '08: Proceedings of the 16th International Symposium on Foundations of Software Engineering*, November 2008.

[8] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In *ICSM '08: Proceedings of the 24th IEEE International Conference on Software Maintenance*, pages 337–345, September 2008.

[9] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced? bias in bug-fix datasets. In *ESEC-FSE '09: Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2009.

[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 *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 518–528, Washington, DC, USA, 2009. IEEE Computer Society.

[11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Investigating information needs to improve cooperation between developers and bug reporters. In *CSCW '10: Proceedings of the ACM Conference on Computer Supported Cooperative Work*, February 2010.

[12] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. In *MSR '06: Proceedings of the International Workshop on Mining Software Repositories*, pages 105–111, 2006.

[13] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1767–1772, 2006.

[14] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11. ACM, 2008.

[15] CNET News.com Staff. Microsoft tests its own 'dog food'. `http://news.zdnet.com/2100-3513_22-130518.html`, 2003.

[16] S. Dowdy, S. Weardon, and D. Chilko. *Statistics for Research, volume 1345 of Wiley Series in Probability and Statistics*. John Wiley & Sons, New Jersey, 2004.

[17] P. J. Guo and D. Engler. Linux kernel developer responses to static analysis bug reports. In *USENIX ATC '09: Proceedings of the 2009 USENIX Annual Technical Conference*, pages 285–292, June 2009.

[18] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

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

[20] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Software Eng.*, 29(6):481–494, 2003.

[21] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles. Towards a simplification of the bug report form in Eclipse. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 145–148. ACM, 2008.

[22] L. Hiew. Assisted detection of duplicate bug reports. Master's thesis, The University of British Columbia, 2006.

[23] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *ASE '07: Proceedings of the twenty-second IEEE/ACM International Conference on Automated Software Engineering*, pages 34–43, 2007.

[24] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 2nd edition, 2000.

[25] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *DSN '08: Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 52–61, 2008.

[26] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *ESEC-FSE '09: Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2009.

[27] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *VL/HCC '08: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 82–85, September 2008.

[28] S. Kim and M. D. Ernst. Which warnings should I fix first? In *ESEC-FSE '07: Proc. 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 45–54. ACM, 2007.

[29] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *VL/HCC '06: Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 127–134, 2006.

[30] D. W. van Liere. How Shallow is a Bug? Open Source Communities as Information Repositories and Solving Software Defects. *SSRN eLibrary*, 2009. `http://ssrn.com/paper=1507233`.

[31] D. W. van Liere. Improving Bugzilla's bug overview list by predicting which bug will get fixed, June 2009. `http://network-labs.org/2009/06/improving-bugzilla's-bug-overview-list-%by-predicting-which-bug-will-get-fixed/`.

[32] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In *ICSM '08: Proc. 24th IEEE International Conference on Software Maintenance*, pages 346–355, Sept 2008.

[33] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002.

[34] 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, 2008.

[35] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. Irwin, 4th edition, 1996.

[36] L. D. Panjer. Predicting Eclipse bug lifetimes. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007. MSR Challenge Contribution.

[37] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 499–510, 2007.

[38] J. R. Ruthruff, J. Penix, J. D. Morgenthaler, S. Elbaum, and G. Rothermel. Predicting accurate and actionable static analysis warnings: an experimental approach. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 341–350. ACM, 2008.

[39] E. Sink. My life as a code economist, November 2005. http://www.ericsink.com/articles/Four_Questions.html.

[40] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5. ACM, 2005.

[41] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 461–470. ACM, 2008.

[42] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007.