# Characterizing and Predicting Which Bugs Get Reopened

Thomas Zimmermann [1] tzimmer@microsoft.com
Nachiappan Nagappan [1] nachin@microsoft.com
Philip J. Guo [2] pg@cs.stanford.edu
Brendan Murphy [3] bmurphy@microsoft.com
[1] Microsoft Research, USA
[2] Stanford University, USA
[3] Microsoft Research, UK

***Abstract*—Fixing bugs is an important part of the software development process. An underlying aspect is the effectiveness of fixes: if a fair number of fixed bugs are reopened, it could indicate instability in the software system. To the best of our knowledge there has been on little prior work on understanding the dynamics of bug reopens. Towards that end, in this paper, we characterize when bug reports are reopened by using the Microsoft Windows operating system project as an empirical case study. Our analysis is based on a mixed-methods approach. First, we categorize the primary reasons for reopens based on a survey of 358 Microsoft employees. We then reinforce these results with a large-scale quantitative study of Windows bug reports, focusing on factors related to bug report edits and relationships between people involved in handling the bug. Finally, we build statistical models to describe the impact of various metrics on reopening bugs ranging from the reputation of the opener to how the bug was found.**

***Keywords*—bug triage; bug reopen; bug report**

## I. INTRODUCTION

A key activity in the software development process is fixing bugs submitted by testers and end users. An important but oft ignored aspect is the bug reopen rate in this process. Bugs can be reopened for a variety of reasons ranging from poorly fixed bugs, incorrectly fixed bugs, new changes that required prior closed bugs to be reopened, bugs reopened due to the identification of the actual cause of prior closed bugs, or better reproducibility of a bug. The process of bug reopens also has its own dynamics given the number of bugs assigned to a developer, the geographical distribution of the people opening the bugs, and the type of bugs.

Understanding bug reopening is of significant interest to the practitioner community in order to

- Characterize actual quality of the bug fixing process
- Identify important issues that are not fixed and later reopened
- Identify areas which need better tool support
- Improved bug triage process
- Plan and estimate the effort for bug fixing activity taking into account the reopen rates

To the best of our knowledge there, has been only little work on studying the bug reopen process in research The closest related work is by Shihab et al. [1] who predict which bugs will be reopened in Eclipse using metrics related to work habits, bug report, bug fix and team as input to a prediction model. Our study perfectly complements the work by Shihab et al. [1] and is significantly different from that work: the goal of our paper is not to predict for each individual bug the likelihood of being reopened, but to characterize the overall reopen process. In order to do so we employ a more foundational approach wherein we first survey a large population of experienced developers on the fundamental reasons for bug reopens and qualitatively analyze the responses. We then assess the reasons for reopens from a quantitative perspective using data from Microsoft Windows. To explain the relationship between multiple factors and bug reopens, we also build a statistical descriptive model—not to predict reopens—but as a way to identify statistically the most important factors affecting bug reopens.

An important factor in our paper (and not addressed by Shihab et al. [1]) is the impact of *distributed (global) software development* and *organizational structure*. Often large projects are developed in a distributed fashion around the world. Is the effectiveness of the bug fixing process affected by organizational and geographic barriers? We address this question with data from two releases of Microsoft Windows. Further, we identify the impact of the *reputation* of the bug opener and the engineer fixing the bug on the likelihood of a reopen. We also observe that *how bugs are found* (such as human review, code analysis tool, system testing, customer found, etc.) has a noticeable impact on bug reopens.

Our overall motivation is to help engineers and scientists understand the bug reopen dynamics in large scale software development. Bug reopens have a significant importance in both open source and commercial software systems. The primary aim is helping engineers plan effort for future work, improve the bug triage process, and identify areas for better training for the employees, which can reduce the number of bug reopens caused by limited technical knowledge. In addition, our paper contributes a (partial) replication of the work by Shihab et al. [1], thus increasing the generality of their results, while at the same time contributing new empirical findings, for example, a detailed list of causes of bug reopens and the relationship between bug reopens and geographic and organizational distance.

This paper is organized as follows. In Section II, we summarize related work and in Section III, we describe the methodology of our study. In Section IV, we identify causes of bug reopens (based on qualitative survey responses) and in Section V and VI we present the results of the quantitative analysis of Windows bug reports. In Section VII, we discuss the threats to validity and we conclude in Section VIII.

## II. RELATED WORK

The work closest to this paper is by Shihab et al. [1] who predicted reopened bugs in the Eclipse project. They used measures from four dimensions—work habits, bug report,

bug fix and team—as input for decision trees (C4.5), which predicted reopened bugs with a precision of 62.9% and a recall of 84.5%. With a *top node analysis* they found that the bug report dimension was most influential. With this paper, we replicate some of the measures by Shihab et al. [1] on the Windows bug database and include new measurements (for example organizational and geographic distance). In addition to the work by Shihab et al. [1], our paper includes a strong qualitative component on the *causes* of bug reopens (identified through survey comments) and also presents complete descriptive models. While Shihab et al. [1] used decision trees which are descriptive too, they only presented the trees aggregated to the top nodes in their paper.

Several other 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, WON'T FIX, DUPLICATE, WORKS FOR ME). Hooimeijer and Weimer [2] 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 [3] 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. [4] observed that bug reports are fixed sooner when they contain stack traces or are easy to read. Anbalagan and Vouk [5] found that the more people are involved a bug, the higher its time-to-resolve. Mockus et al. [6] found that in Apache and Mozilla, bugs with higher priority are fixed faster than bugs with lower priority. Herbsleb and Mockus [7] 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. [8] 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 *when bug reports are reopened.*

Several studies characterized properties of bug reports and their edit activities: Bettenburg et al. [4] characterized what makes a good bug report. Aranda and Venolia [9] examined communication between developers about bug reports at Microsoft to identify common bug fixing coordination patterns. Breu et al. [10] categorized questions asked in open-source bug reports and analyzed response rates and times by category. Bettenburg et al. [11] quantified the amount of additional information in bug duplicates. Jeong et al. [12] analyzed the reassignment of bug reports (called bug tossing) and developed tossing graphs to support bug triaging activities. Ko et al. [13] conducted a linguistic analysis of bug report titles and observed a large degree of regularity. Bertram et al. [14] conducted a qualitative study of issue tracking systems as used by small, collocated software development teams. They found that even in collocated teams, issue trackers are a focal point for communication and coordination. Ko and Chilana [15] quantified the value of contributions by "power users" to open bug reporting in Mozilla. They observed that the primary value comes from recruiting a small pool of talented developers and reporters, and not from the masses. In our own previous work, we characterized which bugs get successfully fixed [16] and factors that lead to reassignments [17]. However, none of these studies characterized and predicted which bugs get reopened.

To improve bug triaging, previous research proposed techniques to semi-automatically assign developers to bug reports [18,19,20], assign locations to bug reports [21], recognize bug duplicates [22,23,24,25], assess the severity of bugs [26], and predict effort for bug reports [27]. With this paper we analyze bug reopens, which allows assessing the effectiveness of many of these triaging activities.

Empirical studies allow us to build a validated body of knowledge in software engineering [28] and are crucial for informing the design of new bug tracking tools. This paper adds a characterization of what bug reports are reopened to that body of knowledge.

## III. METHODOLOGY

We studied reopens of bug report reopens in the context of two versions of the Microsoft Windows operating system project, which we feel is a representative example of a large-scale commercial software project. Both Windows Vista and Windows 7 contain several thousand source code files and 40+ million lines of code, written by more than 2000 software engineers. The findings we present in this paper are derived from three sources related to Windows bug reports: free-response answers from a survey sent to Microsoft employees, a manual examination of randomly-selected bug reports, and a high-level quantitative analysis of the entire Windows bug database.

### *A. Survey Free-Response Answers*

Our primary data source is an online survey we sent in August 2009 to 1,773 Microsoft employees with questions about various aspects of the bug triaging and fixing process. Since we wanted to get the opinions of people well-versed in handling Windows-related bugs, we chose as our survey participants the top 10% of people who have opened, been assigned to, or resolved Windows Vista bugs. 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 analyzed responses to most of the survey questions for another paper [16]; for this paper, we analyzed responses to the following free-response question, which we did *not* explore in our other paper:

> In your experience, what are some reasons why a bug would be **reopened multiple times** before being successfully resolved as *Fixed*? E.g., why wasn't it assigned directly to the person who ended up fixing it?

Response length varied from one phrase (e.g., "bug cause was not initially understood") to long paragraphs. We printed out all 358 responses on index cards for a card sort [29]. Two of the authors independently performed an open card sort and then merged their results into a single taxonomy.

### B. Manual Examination of Bug Reports

Informed by our analysis of survey results, we informally examined the contents of 20 Windows Vista bug reports with reopens, chosen by randomly sampling. The main reason we manually examined selected bug reports was to corroborate the survey respondents' opinions with firsthand observations from the bug reports themselves.

### C. Quantitative Analysis of Bug and Personnel Data

We quantified certain observations to the extent possible by mining data from the Windows bug database and the Microsoft employee personnel database. First, we collected all pre- and post-release bug reports for Windows Vista 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 (2002-2007). We also extracted bug reports from Windows 7, where we used the entire bug database for the development period of Windows 7 (~3 years). For confidentiality reasons, we cannot reveal the exact number of bug reports, but it is at least an order of magnitude larger than datasets used in related work [2]. For each bug report, we extracted a list of edit events that occurred throughout its lifetime. Each event alters one or more of the following fields (fields not relevant to our analysis in this paper have been omitted):

- **State:** Opened, Resolved, or Closed.
- **Opener:** Who opened this bug?
- **Assignee**: Who is now assigned to handle this bug?
- **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).
- **Component path:** Which component is the bug in? e.g., DesktopShell/Navigation/StartMenu
- **Bug type:** What kind of bug is it? e.g., bug in code, specification, documentation, or test suite
- **Bug source:** How was this bug found? e.g., by a customer, an internal Microsoft user, or a system test
- **Resolution status:** How has this bug been resolved? e.g., FIXED, BY DESIGN, NOT REPRODUCIBLE, WON'T FIX. (Null if state is not RESOLVED)

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 that he/she has resolved the bug, its "Resolution status" field is set. After the resolution attempt is approved (usually by the bug opener), the bug is closed. However, the bug might be **reopened** if it has not actually been properly resolved.

To explore the impacts of geographical and organizational distance on bug reopens, we obtained the office location and manager of each employee circa July 2009 from the Microsoft employee personnel database. Thus, we can determine whether two employees worked in the same building, campus, country, or on the same team (i.e., had the same manager). Sometimes people switch locations or teams, but in general Microsoft tries to keep employees in the same location and team during a product cycle [30].

Table 1 Causes of bug reopens.

| | |
|---|---|
| Not FIXED | Related to *Root Cause:*<br>– Bugs difficult to reproduce<br>– Developers misunderstood root cause<br>– Bug had insufficient information<br><br>Related to *Priority*<br>– Priority of the bug increased |
| FIXED | – Regression bugs |
| Process-related | – Process |

## IV. CAUSES OF BUG REOPENS

In the card sort, we identified six different categories that fall into three groups (see also Table 1):

1. Reopened bugs that were resolved as *not* fixed — either related to the root cause or the priority of the bug report. (Section IV.A)
2. Reopened bugs that were resolved as fixed — typically regressions. (Section IV.B)
3. Process-related bug reopens. (Section IV.C)

Reopens related to the *root cause* are mostly about reproducing and understanding the bug. When a bug marked as WON'T FIX is reopened, typically additional steps to reproduce or information has become available that now allows finding the root cause.

Several factors matter for the priority of bug reports [16]: severity (how bad is the bug?), impact (how many people are affected?), effort to fix (how much time?), and the risk of the fix (how likely are regressions? where in the product cycle?). For reopens related to the *priority* of a bug, typically information emerged that increases the priority, for example more customers experienced the bug (larger impact), feedback shows a higher severity, or changing business needs.

### A. Reopened Bugs that Were Resolved as Not Fixed

Bug reports resolved as *not* fixed (for example BY DESIGN, NOT REPRODUCIBLE, WON'T FIX, etc.) can be reopened when (1) developers initially could not properly identify the *root cause* —because a bug is difficult to reproduce, steps to reproduce are incomplete or missing, or simply because of misunderstandings—or (2) developers underestimated the priority of a bug report or its importance changed over time.

**Bugs difficult to reproduce.** The largest category in the card sort corresponds to reopens related to reproducing a bug. There are several reasons why bug reports can be hard to reproduce: incomplete, ambiguous, or complex steps to reproduce; timing related bugs; or Heisenbugs *("a crash that will not occur when under a debugger")*. Typically these bugs get closed as "no repro" and reactivated if they occur more often or if new reproduction details surface.

*These are usually flaky bugs or hard to repro bugs. For example, the developer is not able to reproduce it but the tester is able to.* [1]

*Most commonly when not readily reproducible, and lacks multiple reports. Later on a consistent repro may be found, or more hits may occur, and the bug would be reopened.*

*Bugs which are difficult to reproduce generally get reactivated multiple times. At first, developers will give a simple repro attempt before resolving bugs 'Not repro'. But if the bug opener is able to reproduce the issue again, or perhaps comes up with better repro instructions, then the developer will pay more attention the second time the bug is activated.*

*Intermittent repro's. Often indicative of race conditions and other environmental factors. Lack of repro machine.*

Some bugs are difficult to reproduce in house and only happen on customer machines, which makes it more difficult to verify that bug was fixed correctly.

*Bugs which are difficult to repro eventually need to get timed out and closed. When a repro surfaces, the bug will come back. But then an evaluation of the customer impact might cause it to get won't-fixed and closed, but then an actual customer hits it so we finally go ahead and fix it now that there's real data to substantiate the need for a solution.*

*The bug is hard to reproduce and so the fix was made without being able to fully verify it. A good example is a customer who reports something. We think we see the issue in house and fix that. It turns out we saw something different.*

**Developers misunderstood the root cause.** For some bug reopens, developers initially did not understand the root cause and as a result the bug report was incorrectly closed or fixed. Often root causes are difficult to identify, for example when related to memory leaks:

*The bug is tracking an unidentified symptom and it takes a while to fully root cause. This comes up a lot with memory leaks: there will be an unknown memory leak in a component and the owning team plays whack-a-mole with the code defects to remove memory issues one-by-one. When this team fixes a leak, it's not entirely sure that it fixed the* entirety *of the leaks inside that component, but it's clear that they changed something. So, the bug will be resolved with each change and then reactivated when tests are re-run to isolate the leak further.*

There can also be disagreement between teams about the root cause and who is responsible to fix the bug.

*No clear agreement on the root cause – each team involved thinks the root cause lies in someone else's component*

If the wrong root cause is identified, a different issue might be fixed and the bug report will likely be reopened.

*Another variant is an investigation drawing the wrong conclusions, a fix getting done for some other issue that was found in the investigation, and test not being able to repro properly (possibly an interaction with the fix, or never had a solid repro).*

When the root cause is not yet understood, developers might decide to first fix the symptom and then later revisit the bug to fix the actual root cause.

*Not fixing the root cause and only addressing symptoms. Without root cause understood for the bug a patch/hack can often be done that will then be reactivated.*

A lack of understanding of the root cause is also related to bug duplicates. For example, different bugs with similar symptoms or titles might be accidentally resolved as duplicate and then later reopened.

*I have seen several bugs with very similar symptoms getting resolved as duplicates. In this case the duplicate gets closed but if we later find out it was a different issue it gets reopened. So basically this is not understanding the bug or the impact it might have.*

A related issue is that some bug reports describe multiple defects (either intentionally or accidentally) and only some of them are fixed which requires the report to be reopened.

*The bug is being used as an umbrella for a bunch of smaller bugs, so it bounces open and closed as each issue is found and dealt with.*

*Multiple code defects at the same time cause the same effect (bug). Developer only tries and fixes one cause and tester only verifies that since path initially.*

**Bug report had insufficient information.** Another reason for reopens is that the initial bug report did not have enough information to understand the bug and locate its root cause. Especially bug reports with poor quality fall into this category. They usually get resolved as WON'T FIX and only after additional information is provided, developers can correct the bug. New information that becomes available and leads to bug reopens includes Watson dumps [31], stack traces, time travel tracing [32], screenshots, or detailed environment information (such as hardware, software, and network configuration).

*Poor bug quality: If the bug wasn't described well enough, or not enough diagnostic info was there, the dev will guess and fix *something* in order to make the bug go away. What they fix isn't always what the person who filed the bug ran into.*

*If a bug report does not accurately convey enough information about what is actually wrong (i.e. it describes incorrect behavior but neglects to mention data loss) or if the bug does not convey a dependency (such as another team relying on a fix), a bug may be de-prioritized and resolved without fixing.*

**The priority of the bug increased.** Another reason for reopens is when the severity or impact of a bug has been underestimated and new information becomes available that indicates the higher importance.

*Bugs are closed because one person or triage team believes the bug is not worthy of fixing (ie. too risky, don't care, etc.), but then a few days later a VP or external customer reports the same issue, then the bug has a higher priority.*

*We see one repro, nothing happens for a month so we triage the issue as won't fix, then suddenly we see multiple repros a day and it becomes more critical.*

Bugs can also become more important as business needs, customer scenarios, and management changes. Bugs that are not fixed in one release might be fixed in the next release.

[1] In the following, each italicized, indented paragraph corresponds to a quote from the free-response answers in our survey.

*Other reason is lack of business justification or too late in product cycle; reopened when sufficient justification exists or new cycle begins.*

*People may argue it is too late in the product cycle to fix the bug, so it got resolved as wont-fix […] until the next release.*

Developers, testers, and triage teams can also disagree about the priority.

*One team may feel an issue is critical while the other does not see it as important enough, and instead of carrying a discussion, the bug is bounced around.*

*Disagreement about severity and priority is most often the issue in the bugs I have been dealing with. If the person assigned the bug does not view it as severe enough to warrant a fix (especially as we are nearing release) then every bug has to be fought for by the person who opened it.*

### B. Reopened Bugs that Were Resolved as Fixed

This category includes mostly **regression bugs**, i.e., bugs that were fixed in a previous revision but reappeared in a current version of the system. Several reasons for regressions were mentioned in the survey: integrations in the SCM that incorrectly override a correct bug fix *("branch integration removed the fix"*), an incomplete bug fix, and insufficient testing

*First attempt at fix was flawed in some way, and wasn't caught because of lack of testing or unknown related scenario regression.*

*Issue was fixed, but regresses and bug is reactivated rather than a new bug getting filed (Fixed).*

*This usually happens when fixes aren't fully tested before checked in. So a subsequent less tested fix could break the original fix causing the bug to be reactivated.*

For example developers might have missed a corner case, which only gets apparent in later testing of the system.

*The developer may have overlooked some additional edge cases related to the scenario being tested.*

*I've seen cases in the past where it was thought that a bug was fixed only to find that a corner case had been missed. I've also seen cases where the bug was only being hit due to a timing issue and something changed that affected the timing and the bug disappeared again.*

### C. Process-Related Bug Reopens

Several responses were related to the general **process** of fixing bugs. For example, bugs can be reopened because the tester is validating the fix on the wrong version of binaries because the fix has not yet reached the tester's branch.

*Sometimes bugs are reopened due to a misunderstanding of process. e.g. dev resolves bug when fix is submitted, but tester reactivates because bug still repros (because fix has not yet reached tester).*

*Tester reopens the bug because they do not realize that the bug IS fixed but just not in the build that they are testing on (this happens all the time)*

*Bug is verified fixed in a feature of developer's branch and the fix takes too long to hit the main branch.*

Some responses suggested that bug reopens can also happen when developers do not pay enough attention to the bug report or testers validate the fix insufficiently.

Table 2 The influence of the bug source on bug reopens

| BUG SOURCES | Vista | Win7 |
|---|---|---|
| Reopen rate for all bugs | P | Q |
| Code analysis tools | 0.52·P | 0.73·Q |
| Human review | 0.85·P | 0.66·Q |
| Ad-hoc testing | 0.87·P | 0.99·Q |
| Internal user | 1.12·P | 0.97·Q |
| Component testing | 1.13·P | 0.81·Q |
| System testing | 1.21·P | 1.46·Q |
| Customer | 1.33·P | 1.12·Q |

*I found that some devs don't spend much time trying to reproduce bugs, rather they just push back on the test team. In this case, the bug will be resolved, and re-opened multiple times until the tester either writes a simple repro, or the tester (or maybe willing dev) actually spends the time to debug the issue.*

*Having the bug closed and then reactivated multiple times sounds like a case of the tester doing really bad job. If the bug wasn't fixed – they shouldn't close it but reactivate it back to the developer – and at that time provide an more detailed description of the problem they are seeing – something beyond 'I can still repro'.*

Lastly, while not directly a reason for bug reopens per se, one respondent pointed out that reopens complicate tracking of bugs and changes.

*First of all, I don't like the model where we reactivate bugs that were Fixed but the issue was not resolved. Logically it makes sense, but tracking the thread of the issue through multiple checkins & reactivates can be hell if it happens more than once or twice. I would prefer a model where once a checkin has been made for a bug, that bug is done! New issues, or issues that linger despite a previous fix, should/ would be tracked in a new bug.*

## V. INFLUENCES ON BUG REOPENS

We now present several factors related to bug reports and people that affect the likelihood of a bug being reopened.

### A. Does the Source of a Bug (How it Was Found) Influence the Likelihood of Bug Reopens?

We observed that bugs from certain sources were more likely to be reopened than other bugs. To quantify this effect, we grouped bugs based on how they were found. Then we calculated the percent of bugs in each group that were reopened. In Table 2, we report the reopen ratios relative to the baseline percentages P and Q, which are the reopen rates for all bugs in Windows Vista and Windows 7 respectively.

In Table 2, we observe that bugs found by *code analysis* tools or during *human reviews* are *less likely* to be reopened (0.52~0.73 times for code analysis tools and 0.66~0.85 times for human review). Possible reasons might be that some bugs found by code analysis tools are easy fixes; also code analysis tools do not argue that a bug should be fixed, once it has been closed as WON'T FIX. The group human review consists of bugs found during code, design, spec, or security reviews. In previous work we found that bugs identified by human

review are more likely to be fixed [16], thus we expect that there is less need to reopen unfixed bugs. Furthermore, human review and code analysis bugs are easy to triage and require no separate reproduction steps.

Table 2 shows that bugs found by *customers* and during *system testing* (e.g., integration, build, and stress tests) are *more likely* to be reopened (1.26~1.46 times for system testing and 1.12~1.33 for customer bugs). Section IV discusses several reasons for this observation: bugs from these sources are often more complex and more difficult to reproduce and thus more difficult to fix. Most users are not trained to write methodical bug reports like for example professional testers are. Once new information becomes available these bugs get reopened.

### B. *Does the Reputation of the Opener and First Assignee Influence the Likelihood of Bug Reopens?*

We found that a bug opened by someone who has been successful in getting his/her bugs fixed in the past (i.e., has a better reputation with respect to bug reporting) are less likely to be reopened—surprisingly also bugs by highly *un*successful people are less likely to be reopened.

We quantify the reputation of bug openers using the same metric as Hooimeijer and Weimer [2], which is based on success rate:

$$\text{bug opener reputation} = \frac{|\text{Opened} \cap \text{Fixed}|}{|\text{Opened}| + 1}$$

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 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 (left plot) and Windows 7 (right plot) bugs by ranges of opener reputations and plotted the percentage of bugs in each group that were reopened. The leftmost two points in each plot 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.

Looking at the bug reports in Windows Vista (left plot), starting 0.20 there is a consistent monotonic decrease in bug-reopen likelihood as the opener reputation increases. Interestingly, bugs opened by first-timers ("First bug" point) and people with low reputations had a lower reopen likelihood

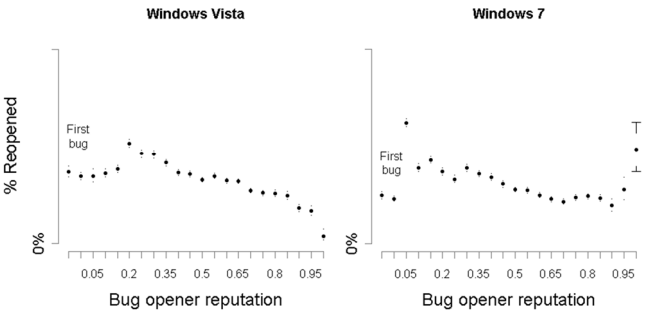


Figure 1 Percent of reopened Windows Vista (left) and Windows 7 bugs (right) vs. bug opener reputation (rounded up to nearest 0.05)."First bug" represents all bugs whose opener had never opened a bug before the current one. The y-axis is unlabeled for confidentiality.

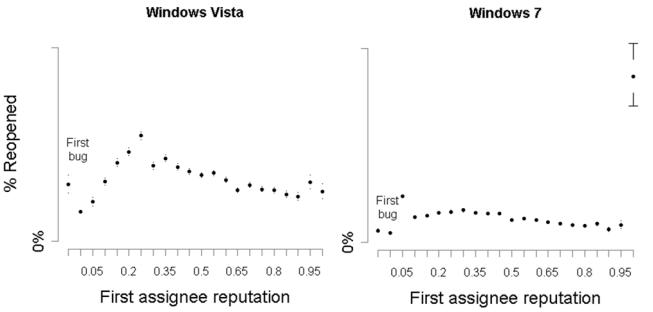


Figure 2 Percent of reopened Windows Vista (left) and Windows 7 bugs (right) vs. first assignee reputation (rounded up to nearest 0.05)."First bug" represents all bugs whose opener had never opened a bug before the current one. The y-axis is unlabeled for confidentiality.

than bugs opened by people with low reputations between 0.20 and 0.40. All differences between points are statistically significant since their 95% confidence intervals for binomial probabilities never overlap [33]. In fact, most confidence intervals are negligible, so they are invisible in Figure 1.

We observed a similar, although less distinct, effect for Windows 7 in the right plot of Figure 1. Here, we have the lowest reopen rates actually for people who opened their first bug report or never opened a bug that was successfully fixed (zero reputation). Note that in Windows 7 only few people had a reputation greater than 0.90, which explains the wide confidence intervals. We believe that the low reopen rates for highly successful and highly unsuccessful people are because they are less likely to argue for a bug. People with high reputation will have the experience to decide whether a reopen is worth the effort, while people with low reputation might lack the confidence to argue that a bug report should be fixed.

When we repeated these calculations for the reputations of the first person who was assigned to each bug, the trends were nearly identical (see Figure 2). This result shows that certain people are more effective at either fixing bugs or reassigning bugs to others who can fix them, that is, their bugs are less likely to be reopened after they have been closed.

### C. Does Organizational and Geographic Distance Influence the Likelihood of Bug Reopens?

We found bug reports were more likely to be reopened when *initially* assigned to someone in a different team or geographical location as the bug opener. We also observed that bugs that were assigned *at some point in time* back to the opener or his/her team were less likely to be reopened—at first glance this seems to be counterintuitive; however, when bugs are reopened, they are often assigned back to the opener or his/her team to solicit additional information. In other words, assignments back to the bug opener (at some point) are indicative of problems and reopens..

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 reopened. In Table 3, we report ratios relative to the anonymized baseline percentages X, Y, and Z for Windows Vista and R, S, and T for Windows 7. For instance, Vista bug reports opened by and initially assigned to people with different managers are 1.37 times as likely to be reopened as those opened and initially assigned to the same person (shown in bold near the top of Table 3).

For the ***initial assignment***, bugs opened by and assigned to the same person are the least likely to be reopened (the X baseline for Vista). Here the opener is often a developer who wants to and is able to fix his/her own bug, which means that many of the causes for bug reopens do not apply in the situation (for example, the developer very likely could already reproduce the bug and has an idea how to successfully fix it). The primary cause of reopens for such self-assigned bugs is regressions or incorrect/incomplete fixes. Bugs assigned to someone in the same team or building have a slightly higher reopen rate in Vista (1.13 and 1.27 times, respectively) and a slightly lower reopen rate in Windows 7 (0.96 and 0.93 times, respectively). These colleagues can easily talk face-to-face to resolve ambiguities in bug reports and to hold each other accountable. As a result reopen rates for within-team/ within-building bugs are comparable to self-assigned bugs.

However, if bugs are assigned to people who work in different buildings or countries, then there is greater overhead in communication, which leads to more bug reopens, both in Windows Vista (up to 1.52 times) and in Windows 7 (up to 1.14 times). In our earlier work on which bugs get fixed [16], a survey respondent cited *"poor communication, language barrier problems with other countries"* as hindrances in a free-response question about what factors affect bug fixes. 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. Herbsleb and Mockus found that modification requests, which include bug reports, took longer to resolve when development was globally distributed [7]. Our findings supplement theirs by showing that bug reports *initially* assigned across teams, buildings, and countries are more likely to be reopened. This might suggest that sections of software should be outsourced (e.g., the development and testing of particular parts of the architecture) rather than functions (e.g., outsource the testing effort).

Table 3 The influence of organizational (team structure, temporary employees) and geographical factors on bug reopens

| INITIAL ASSIGNMENT | Vista | Win7 |
|---|---|---|
| Opened by and *initially* assigned to … | | |
| … the same person | X | R |
| … someone with the same manager | 1.13·X | 0.96·R |
| … someone with a different manager | **1.37·X** | 1.07·R |
| Opened by and *initially* assigned to … | | |
| … the same person | X | R |
| … someone in the same building | 1.27·X | 0.93·R |
| … someone in a different building but in the same country | 1.45·X | 1.00·R |
| … someone in a different country | 1.52·X | 1.14·R |

| ASSIGNMENT AT SOME POINT IN TIME | Vista | Win7 |
|---|---|---|
| Assigned to opener at *some point in time* | Y | S |
| Never assigned to opener, but assigned to someone with the same manager as opener | 0.54·Y | 0.39·S |
| Never assigned to anyone with same manager | 0.27·Y | 0.34·S |
| Assigned to opener at *some point in time* | Y | S |
| Never assigned to opener, but assigned to someone in the same building | 0.41·Y | 0.37·S |
| Never assigned to anyone in same building, but assigned to someone in the same country | 0.31·Y | 0.43·S |
| Never assigned to anyone in the same country | 0.20·Y | 0.20·S |

| TEMPORARY EMPLOYEES | Vista | Win7 |
|---|---|---|
| Opened by a permanent employee | Z | T |
| Opened by a temporary employee | 1.26·Z | 1.11·T |
| Initially assigned to temp. employee | 1.18·Z | 0.82·T |
| Assigned to temp. employee at any point | 1.62·Z | 1.35·T |

Things change however when bugs are reassigned back to the bug opener or his/her team at ***some point in time*** of the bug's lifecycle. In Table 3, bugs opened by and assigned to the same person at some point in time actually have the highest reopen rate (baseline Y for Vista). The lowest reopen rates are for bugs that never return to the same team or the same country (0.27 and 0.20 times respectively). The reopen rates in these situations are similar for Windows 7 (0.34 and 0.20 times respectively, with baseline Y). There are two reasons for this phenomenon: (1) as mentioned before, when bugs are reopened, they are often assigned back to the opener or his/her team to solicit additional information and thus bug reopens and assignments back to the opening team (at some point) are related events, and (2) bugs that never return to the opener's team (or building, country) might not be critical for the opener and thus they are less likely to reopen the bug when it gets resolved as say WON'T FIX.

At another extreme, ***temporary employees*** (for example, contractors or interns) have lower reputation and fewer relationships with developers, so their bug reports might not be taken as seriously as those from core employees. Table 3 shows that bugs involving temporary employees are more likely to be reopened (up to 1.62 times) compared to bugs by

permanent employees (baselines Z and T) with the exception of Windows 7 bugs that were initially assigned to temporary employees (0.82 times) .

## VI. Descriptive Statistical Model

A problem that often arises when presenting a series of single-variable correlations (as we've done in the previous section with factors that correlate with reopened bugs) is that their effects might be cross-correlated, thereby diminishing their validity. To show that the factors have *independent effects*, we built a logistic regression model [34].

### A. Model Building

A logistic regression model aims to predict the probability of an event occurring (e.g., will this bug be reopened?) 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).

Using our entire bug dataset as training data, we built three models to describe the different aspects of bugs in Windows Vista and Windows 7 (see Table 4):

1. *Probability that a bug will be* ***reopened****.*
2. *Probability that a bug will be* ***fixed after the bug has been reopened.***
3. *Probability that a bug will be* ***fixed****.* We addressed this question in previous work [16]. However, to facilitate comparison we include this model in this paper. (Note that the earlier work only listed the coefficients for Windows Vista because of space reasons.)

By comparing the second model (likelihood of fix for *reopened* bugs) and the third model (for *all* bugs) we can identify factors that more strongly influence the chances of reopened bugs getting fixed than in the general case.

To build the models, we chose the same factors (explanatory variables) as in our previous work on which bugs get fixed in Windows [16] to enable comparison across papers. For the models that describe the likelihood of reopens, we exclude the factor "Number of re-opens" (as indicated by the "n/a"). For each factor, we further tested that its coefficient is statistically significant at $p<0.001$. Almost all coefficients were significant; we indicate coefficients that were not significant with "n.s." in Table 4.

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 reopened, since it uses factors that are not available at bug opening time (such as the number of edits or building changes).

### B. Meaning of Logistic Regression Coefficients

One benefit of using logistic regression over other types of models (e.g., support vector machines) is that its parameters (i.e., the coefficients in Table 4) have intuitive meanings.

**Numerical and Boolean factors:** The sign of each coefficient is its *direction* of correlation with the probability of an event (in our case either successfully fixed or bug reopened). For example, bug opener reputation is negatively correlated with bug reopens in Windows Vista, so its coefficient is negative (–0.266). The magnitude of each coefficient is proportional to how much a one-unit change in the factor affects the probability (for a Boolean factor, FALSE to TRUE is the only one-unit change). For details on transforming coefficients into exact probabilities, see Hosmer and Lemeshow [35].

In general, it's hard to compare coefficient magnitudes across factors since units 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"'. For bug reopens in Vista the coefficient of the former (0.721) is larger than that of the latter (0.468), which means that having the same manager at any point in the bug life cycle has a larger positive effect on bug reopen rates 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 4 as indicated by the symbol ♣). 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 "Code analysis tool" is lower than that for "Component testing". This means that bugs in the former category are less likely to be reopened than bugs in the latter.

**Intercept:** In addition to coefficients, logistic regression also produces a numerical intercept, which here represents the base probability given that all factors are zero. However, we cannot report its value due to confidentiality reasons.

### C. Interpreting the Descriptive Model "Reopen"

The following factors are *positively correlated* with bug reopens, as indicated by the corresponding coefficients, which are positive: whether the opener was a temporary employee (only Vista), whether the opener and any assignee had the same manager or worked in the same building, whether severity was upgraded (only Vista), and number of bug report editors and assignee buildings. The following factors are *negatively correlated* with bug reopen probability, so the coefficients are negative: reputation of the bug opener (both versions of Windows) and first assignee (only Windows 7), the initial severity level, and the numbers of component path changes.

As shown in Section V, bugs from different sources vary in how often they are reopened. Not all sources however are statistically significant in the model, e.g., human review and internal user were not significant in Windows Vista. Recall that for categorical factors the rankings of coefficients have to be analyzed as in our case they are all relative to the factor "Ad-hoc testing". The rankings mostly match Table 4 in Section V, except that in the regression model ad-hoc testing increased its rank from #3 to #2 and customer found bugs decreased the rank from #2 to #7. This means that when looking at multiple factors, bugs found by customers are less likely to be reopened in Windows 7, while in Vista they are more likely to be reopened.

Table 4 Descriptive logistic regression models for (1) bug reopen rate, (2) bug-fix probability for reopened bugs, and (3) bug-fix probability for all bugs. Models are trained Windows Vista and Windows 7 bugs respectively. The factor labeled ♣ folds into the intercept term, which is omitted for confidentiality.

| | COEFFICIENTS (Windows Vista) | | | COEFFICIENTS (Windows 7) | | |
|---|---|---|---|---|---|---|
| FACTOR | Reopen | Fixed When Reopened | Fixed [16] | Reopen | Fixed When Reopened | Fixed |
| Bug source (7 categories) | | | | | | |
| Human review | n.s. | 0.377 | 0.511 | -0.343 | 0.529 | 0.770 |
| Code analysis tool | -0.503 | n.s. | 0.357 | -0.291 | 0.884 | 0.349 |
| Component testing | 0.238 | -0.160 | 0.065 | -0.116 | 0.406 | 0.488 |
| Ad-hoc testing | ♣ | ♣ | ♣ | ♣ | ♣ | ♣ |
| System testing | 0.204 | n.s. | -0.129 | 0.182 | -0.342 | -0.040 |
| Customer ★ | 0.239 | -0.498 | -0.347 | -0.466 | -0.511 | -0.427 |
| Internal user ★ | n.s. | -0.465 | -0.454 | -0.611 | -0.398 | -0.723 |
| (★Relatively fewer bugs from sources marked as ★ were fixed, due to many duplicate bug reports and difficulty of reproducing bugs reported by field users) | | | | | | |
| Reputation of bug opener | -0.266 | 1.632 | 2.193 | -0.948 | 1.601 | 2.480 |
| Reputation of 1st assignee | n.s. | 1.651 | 2.463 | -0.697 | 1.589 | 2.407 |
| Opened by temporary employee | 0.178 | -0.144 | -0.125 | n.s. | -0.403 | -0.260 |
| Initial severity level | 0.127 | n.s. | 0.033 | 0.081 | 0.383 | 0.202 |
| Severity upgraded? | 0.331 | n.s. | 0.256 | n.s. | 0.463 | 0.300 |
| Opener / any assignee same manager? | 0.721 | n.s. | 0.676 | 0.149 | n.s. | n.s. |
| Opener / any assignee same building? | 0.468 | n.s. | 0.270 | 0.376 | n.s. | 0.493 |
| Num. editors | 0.236 | 0.127 | 0.240 | 0.236 | 0.125 | 0.289 |
| Num. assignee building | 0.090 | -0.213 | -0.257 | 0.101 | -0.111 | -0.145 |
| Num. component path changes | -0.160 | -0.162 | -0.232 | -0.053 | -0.135 | -0.214 |
| Num. re-opens | n/a | n/a | -0.135 | n/a | n/a | 0.024 |

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

**Severity:** Bugs opened with a higher initial severity value (in the range of 1–4) are more likely to be reopened, as reflected by positive coefficients in Vista and Windows 7. In Vista, an upgrade in the severity value is also linked to higher reopen likelihood. As discussed in Section IV, a higher severity is one of the main causes of bug reopens.

**Num. component path changes:** Bugs with more component path changes are less likely to be reopened. If people edit a bug report to change its component path, then that might be a sign that they have spent more time on locating the root cause of a bug; thus reopens might be less likely.

### *D. Interpreting the Model "Fixed When Reopened"*

In previous work [16], we identified factors which impact the likelihood of bugs getting fixed in general (see columns "Fixed" in Table 4). For this paper, we built a new model to describe the factors which impact the likelihood of *reopened* bugs getting fixed (see columns "Fixed When Reopened").

By comparing the models we observe that the direction of the effects (indicated by the sign of coefficients) remains the same. However, for some factors the effect decreases; for example, the coefficients for reputation drop from >2.193 in the general case to ~1.600 in the reopened case. Some factors even become statistically insignificant, for example bug found by code analysis tools or during system test (Vista only), factors related to severity (Vista only) or factors related to team structure or geographic location. This suggests that once a bug report has been reopened, these factors are not as important anymore to decide about its final outcome.

## VII. THREATS TO VALIDTY

**Internal validity:** We primarily use the bug repository as the information source. If the bug repository has some error then it will be reflected in our study. However, it is unlikely that the bug repository will miss reopen data or other bug information as Microsoft engineers primarily use this information to track all open bugs and it is not possible for anyone to maintain separate information. Also the study was done after the fact and none of the engineers have any motivation to influence our results in either way. Also, from the survey perspective all the authors were part of Microsoft Research, a parallel organization not associated with any product group in Microsoft. Hence the engineers had no motivation to answer our questions in any specific way/influence the results in any way.

**External validity:** Our study was performed on using Microsoft engineers and Windows bugs as a case study. Drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables [28]. For this reason, we cannot assume that the results will gener-

alize outside of Microsoft or Windows. But there is nothing specific or different in this case study which prevents it from replication in the open source domain. Replications of these studies in different contexts will help generalizing these results and build an empirical body of knowledge.

## VIII. CONCLUSION

Bugs being reopened after being closed are part of the software development process. In this paper we characterized the bug reopen process using a mixed methods approach: we qualitatively identified causes for bug reopens based on the survey responses of 358 Microsoft engineers and performed a quantitative analysis using bug reports from the Windows operating system to assess the impact of the various factors. The findings highlight areas for process improvement: How can we reduce the complexity of branching, which allowed bugs to be 'verified' in the wrong branch and leading to bug reopens? How could the process be changed to allow for a better assignment of initial priorities so that this factor of reopens could be reduced?

**Acknowledgments.** Thanks to the Microsoft Windows team for their help in understanding the data. Philip Guo performed this work during a summer internship and a visit to Microsoft Research.

## REFERENCES

[1] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W.M., Ohira, M., Adams, B., Hassan, A.E., and Matsumoto, K.-i. Predicting Re-opened Bugs: A Case Study on the Eclipse Project. In *Proceedings of the 17th Working Conference on Reverse Engineering* (2010), 249-258.

[2] Hooimeijer, P. and Weimer, W. Modeling bug report quality. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering* (2007), 34-43.

[3] Panjer, L.D. Predicting Eclipse Bug Lifetimes. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories* (2007).

[4] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. What Makes a Good Bug Report? In *FSE '08: Proceedings of the 16th International Symposium on Foundations of Software Engineering* (November 2008).

[5] Anbalagan, P. and Vouk, M. On Predicting the Time taken to Correct Bugs in Open Source Projects (short paper). In *ICSM '09: Proceedings of the International Conference on Software Maintenance* (September 2009).

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

[7] Herbsleb, J.D. and Mockus, A. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Trans. Software Eng.*, 29 (2003), 481-494.

[8] Cataldo, M., Herbsleb, J.D., and Carley, K.M. 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* (2008), ACM, 2--11.

[9] Aranda, J. and Venolia, G. 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).

[10] Breu, S., Premraj, R., Sillito, J., and Zimmermann, T. 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).

[11] Bettenburg, N., Premraj, R., Zimmermann, T., and Kim, S. Duplicate Bug Reports Considered Harmful. Really? In *ICSM '08: Proceedings of the 24th IEEE International Conference on Software Maintenance* (September 2008), 337--345.

[12] Jeong, G., Kim, S., and Zimmermann, T. 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).

[13] Ko, A.J., Myers, B.A., and Chau, D.H. 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* (2006), 127-134.

[14] Bertram, D., Voida, A., Greenberg, S., and Walker, R. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *CSCW '10: Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work* (2010), 291-300.

[15] Ko, A.J. and Chilana, P.K. How power users help and hinder open bug reporting.. In *CHI '10: Proceedings of the 28th International Conference on Human Factors in Computing Systems* (2010), 1665-1674.

[16] Guo, P.J., Zimmermann, T., Nagappan, N., and Murphy, B. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (2010), 495-504.

[17] Guo, P.J., Zimmermann, T., Nagappan, N., and Murphy, B. "Not my bug!" and other reasons for software bug report reassignments. In *CSCW '11: Proceedings of the 2011 ACM Conference on Computer Supported Cooperative Work* (2011), 395-404.

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

[19] Anvik, J. and Murphy, G. Reducing the Effort of Bug Report Triage: Recommenders for Development-oriented Decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*.

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

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

[22] Hiew, L. *Assisted detection of duplicate bug reports*. , 2006. The University of British Columbia.

[23] Runeson, P., Alexandersson, M., and Nyholm, O. Detection of Duplicate Defect Reports Using Natural Language Processing. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering* (2007), 499--510.

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

[25] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. 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* (May 2008).

[26] Menzies, T. and Marcus, A. Automated severity assessment of software defect reports. In *ICSM '08: Proceedings of the 24th IEEE International Conference on Software Maintenance* (September 2008), 346-355.

[27] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. How Long Will It Take to Fix This Bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories* (2007).

[28] Basili, V.R., Shull, F., and Lanubile, F. Building Knowledge through Families of Experiments. *IEEE Trans. Software Eng.*, 25 (1999), 456-473.

[29] Barker, I. *What is information architecture?* , 2005. KM Column, http://www.steptwo.com.au.

[30] Bird, C., Nagappan, N., Devanbu, P.T., Gall, H., and Murphy, B. Does distributed development affect software quality? An empirical case study of Windows Vista. In *Proceedings of the 31st International Conference on Software Engineering* (2009), 518-528.

[31] Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Orgovan, V., Nichols, G., Grant, D., Loihle, G., and Hunt, G. Debugging in the (Very) Large: Ten Years of Implementation and Experience. In *SOSP '09: Proceedings of the Symposium on Operating Systems Principles* (2009).

[32] Cheshire, J. *Image or ImageButton without ImageUrl Causes HTTP GET for Default Document.*. http://blogs.msdn.com/b/jamesche/archive/2009/01/28/image-or-imagebutton-without-imageurl-causes-http-get-for-default-document.aspx.