2009

# How Shallow is a Bug? Why Open Source Communities Shorten the Repair Time of Software Defects

Diederik W. van Liere
*Erasmus University of Rotterdam*, diederik.vanliere@rotman.utoronto.ca

# HOW SHALLOW IS A BUG? WHY OPEN SOURCE COMMUNITIES SHORTEN THE REPAIR TIME OF SOFTWARE DEFECTS

*Completed Research Paper*

**Diederik W. van Liere**

University of Toronto
Rotterdam School of Management, Erasmus University
diederik.vanliere@rotman.utoronto.ca

## Abstract

*A central tenet of the open source software development methodology is that the community of users and developers is instrumental in improving the quality of software. Using a 10-year longitudinal dataset from the Firefox community, I investigate how the size of a community in terms of bug reporters and software developers, the social networks of developers and the quality of user contributions influence the time needed to repair software defects. The results show that a large open source community in terms of bug reporters reduces the time needed to resolve a defect while the addition of new software developers to an open source community takes away resources to fix bugs and increase the time needed to resolve a defect. In addition, software developers occupying dense network positions need less time to solve a bug. Finally, user contributions are beneficial when bugs are lively discussed but there is no support for the prediction that the experience of the bug reporter or the quality of the bug report reduces the time needed to solve a software defect.*

**Keywords:** Open source software, social networks, software development, communities

## Introduction

Few activities of writing software can be as challenging and frustrating as debugging software defects (Eisenstadt 1997). Software defects, or more colloquial referred to as bugs, exist since the invention of the computer. IBM software developers seem to have coined the words bug and debugging in 1944, but other sources indicate that the terms were also used during the time of Edison (late 19th century) in the context of electrical engineering (Kidwell 1998). A bug refers to a logical error in the software source code that results in malfunctioning of functionality. A bug can be classified either as a defect being the non-fulfillment of intended usage requirements or as a nonconformity that refers to the non-fulfillment of specified requirements (ISO 1996). Software quality is a central concern to IS scholars (Banker and Kauffman 2004) and as bugs influence the quality of software it is worthwhile to study effective debugging practices.

Detecting and resolving bugs is becoming more important as companies' use of information systems and off-the-shelf software increases. Many companies increasingly implement software-based business processes and rely on extensive software applications to support them in their daily activities (Butler and Gray 2006). In addition, the IT environment of many companies is highly heterogeneous and often characterized by different operating systems, different networks, and different versions of the same applications that make software more error prone. Finally, malfunctioning software has a detrimental effect on the perceived ease of use which is an important factor that determines user acceptance of a technology (Adams et al. 1992; Venkatesh et al. 2003). In these circumstances, software defects can have adverse consequences for the day-to-day operations of firms. Software defects create significant costs (Banker et al. 1991), can lengthen the time-to-market (Harter et al. 2000), and in some cases can make software developers legally liable. The following case is an illustration of these adverse consequences:

> *"The Internal Revenue Service gave away $318 million in improper refunds this year because a computer program that screens tax returns for fraud was not working. The IRS had contracted with Computer Sciences Corp. to update the program, but the contractor could not produce a working program by the deadline. The old program could not be put back into operation in time for the spring 2006 tax-filing deadline". Washington Post (September 2, 2006)*

Traditionally, the debate about software quality has been framed as two competing views: the "quality is free" view as postulated by Crosby (1979) and the "quality is an investment" view as suggested by Slaughter et al. (1998). Crosby argued that investments in software quality reduce the total software development costs, as software prevention costs are smaller than the rework costs associated with fixing defects. Proponents of the "quality is an investment" perspective point at the decreasing returns to investments in software quality (Slaughter et al. 1998) and the increased development time (Harter et al. 2000). However, both views underscore the importance of formal approaches in improving software quality such as design reviews, code inspection, software testing professionals, and support documentation (Cusumano and Kemerer 1990; Slaughter et al. 1998).

Recently, the rise of the open source movement popularized a third view of improving software quality. A central tenet of the open source software (OSS) philosophy, and popularized by Linus Thorvalds, is that "given enough eyeballs, all bugs are shallow" (Raymond 1999). The assertion is that developers and users scrutinize, review, and improve the software by making the source code freely accessible and redistributable that ultimately leads to software programs with fewer bugs and hence higher quality. This view, which I coin "quality is community-based", is based on the premise that the community of users and developers is instrumental in improving the quality of the software code (Raymond, 1999). While fixing bugs and improving the software quality is of paramount importance to the "quality is free", "quality is an investment", and "quality is community-based" software development methodologies, the methods employed differ starkly. The first two methodologies do not mobilize end-users in a structural way to improve software quality. Instead, software quality needs to be managed top-down and developers need to be trained and educated about the importance of quality (Henderson and Lee 1992; Ravichandran and Rai 2000). In contrast, the input from end-users in improving software quality and solving bugs is a defining characteristic of open source software development. Raymond (1999: 6) summarizes this view as "treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging".

Open source software development, which is often organized as a globally distributed team (Olivera et al. 2008), does not rely on formal development practices such as detailed system level designs and a sequential implementation phases (Vixie 1999). Rather, most open source communities organize themselves as a loosely coupled network consisting of a core team of developers and a swarm of volunteers who can freely enter and exit the community. The design of the software architecture is often emergent, continuously redesigned, and regularly released. The open source software development methodology is decidedly different compared with studies on globally distributed teams that emphasizes management control (Henderson and Lee 1992) and formalized designs (Mockus et al. 2002). Still, anecdotal evidence suggests that the quality of open source software is at least on par with proprietary software (Mockus et al. 2002).

IS scholars have studied the open source phenomena quite extensively in recent years (Fitzgerald 2006; Lakhani and von Hippel 2003; Roberts et al. 2006; Stewart et al. 2006; von Hippel and von Krogh 2003). In particular, the motives of software developers to join an open source community (Hertel et al. 2003), reasons to share software code (Olivera et al. 2008) and knowledge (Wasko and Faraj 2005) and the factors that determine the success of an open source community (Stewart et al. 2006) have received considerable attention. But the most important assertion of the open source philosophy, that the user community and developers is effective in improving the quality of software, has not been scrutinized yet.

The contribution of this paper is to study the merits of the OSS tenet that its community helps shortening software defect repair time by investigating the mechanisms an open source community uses in shortening the time needed to fix a bug. Understanding the mechanisms by which software defects are resolved can help reduce costs and shorten time to market, goals that are relevant for practitioners, but it also sheds new light on the factors that contribute to the success of an open source project. The research question I seek to answer is “Does an open source community reduce the time needed to repair a software defect?” As the quality of software is improved by fixing one bug at the time, I move the unit of analysis from the product level (Harter et al. 2000) to an individual software defect. In particular, I will focus on three aspects of an open source community that could shorten the time needed to resolve a defect: the size of the community in terms of developers and active members, the social networks of the developers and the contributions submitted by active members such as bug reports and discussions.

I proceed as follows. First, I highlight the mechanisms by which an open source community can shorten the repair time of a bug. Drawing from the literature on open source communities (Hertel et al. 2003; Lerner and Tirole 2002), user driven-innovation (von Hippel 2007; von Hippel and von Krogh 2003) and social networks (Burt 1992; Burt 2005), I derive a number of testable hypotheses. Next, I introduce the Firefox browser as my research setting and explain my data collection effort and variable construction. Then I present my results and conclude with a discussion, limitations, and future research.

## Theory

Before I highlight how an open source community fixes bugs and improves quality of the software, I start with a definition of open source software and the different roles within an open source community. I define open source software (OSS) as a project aiming at developing software that gives users certain inalienable rights such as freedom to distribute the source code, freedom to change and improve the source code and freedom to use the source code for any purpose[1]. Source code is the human-readable form of software that a compiler translates in machine-readable binary code. Open source refers to a community-based model of innovation (von Hippel and von Krogh 2003) which is made possible by a view on intellectual property that stresses freedom for the user. An open source community has different types of memberships. First, there are end-users who use the software in their daily activities but who do not actively contribute to the community. Next, there are community members or bug reporters (I will use these terms interchangeably) who are people that use the software and help developers in different ways including filing bug reports and joining discussions about possible causes and solutions (Bagozzi and Dholakia 2006). In a typical open source community, the number of bug reporters is a small percentage of the total number of users. Finally, developers are people who contribute source code to a software project. In the remainder of this paper, I focus exclusively on how a community surrounding a software project helps improving the quality of the code by reducing the repair time of bugs.

Open source software development is based on a community of both developers and bug reporters that jointly build and improve software. I define the community broadly as the collection of developers and bug reporters that have organized themselves around a particular software project. Previous research has identified different roles for

[1] This definition is based on the free software definition available at http://www.gnu.org/philosophy/free-sw.html and the open source definition available at http://www.opensource.org/docs/osd. My definition covers the most important rights for users but it is not exhaustive and there are philosophical differences between open source software and free software that are beyond the scope of this paper.

members of a community but for the purpose of this study, I will focus on two main roles: developers and bug reporters[2].

The open source community, the networks of the software developers, and the contributions of the community members reduce the repair time for a bug. First, I theorize at the level of the community how characteristics of the community affect the time needed to resolve a bug. Next, I theorize from the point of view of an individual developer under what circumstances s/he is more effective in solving bugs and finally I theorize about the characteristics of the user contributions that help shorten the repair time of a software defect.

## *How the Open Source Community Shortens the Repair Time of a Bug*

Community members are instrumental in improving the quality by stress testing the software in ways that cannot be imagined during unit testing, formal code reviews, and beta tests. Detecting a software defect in an open source community goes through a funnel that consists of four stages. The first stage is that the defective code must be made accessible, either as a beta or stable release, and be part of a functionality that is accessible by a user. The second stage is that a user triggers the defective code. The third stage is that someone, usually an end-user, notices the anomaly and is aware that the software code does not generate the expected output. Finally, the fourth stage is the act of reporting the erroneous or unintended result back to the open source community. Each stage of the funnel shrinks the number of potential users who can report the software defect; however, the likelihood of detecting a defect increases as you go deeper in the funnel. This is the reason why the size of a community, in terms of bug reporters, is crucial for shortening the time needed to repair a bug. The final stage of the funnel should be reached with at least one end-user if the defect is going to be repaired. Larger communities with more active community members are more likely to reach this stage. Therefore, I hypothesize:

> **Hypothesis 1 (Bug Reporter)**: The repair time of a software defect decreases as an open source community grows in the number of bug reporters.

A well-known historical problem of managing software development, and by extension fixing bugs, is that adding more developers to a project does not reduce the time needed to finish the project. This is sometimes referred to as Brooke's law (1975) and states "adding man power to a late software project makes it later". The reason is that communication between developers grows exponentially while the team grows linearly. Developers spend less time working on the code and more time communicating. Open source communities are able to escape from Brooke's law by parallelizing the development and debugging effort as there is no formal authority within an open source community who determines the resource allocation. The parallelization of software development means that the tasks can be partitioned in a scalable manner. Adding a developer to an open source community does not harm the productivity of the other developers and hence, I hypothesize:

> **Hypothesis 2 (Active Developer)**: The repair time of a software defect decreases as an open source community grows in the number of active software developers.

## *How the Community of Software Developers Shorten the Repair Time of a Bug*

Writing software code is a complex activity with many tacit aspects. Software development can be compared with writing an academic research paper: there are many ways of conducting and writing academic research and not all of them will result in a high quality publication. One cause is that some weaknesses in an academic paper will only surface very late in the research process at which stage they have become irrevocable.

The same is true for software development. There are many different ways for implementing certain functionalities, some of which are flawed at the start of implementation while others will have weaknesses that only become apparent when the software code is interacting with other modules or functionalities. At this point, it usually becomes increasingly costly to repair due to dependencies that are often characteristic for software development (MacCormack et al. 2006). Formal methodologies can help a developer in writing software code of higher quality

[2] Albeit, one can distinguish more fine-grained roles within an open source community such as "gatekeepers" (von Krogh et al. 2003) and "core developers" (Dinh-Trong and Bieman 2004) such a more fine-grained distinction does not give additional insight in how an open source community repairs software defects although it may be of importance to how the workload is divided.

but informal information and knowledge sharing about best practices and tips is as important for a developer to improve the quality of the software code (Hoopes and Postrel 1999).

The community of software developers can be conceptualized as a social network (Wellman et al. 1996). Software developers are the nodes of this network and developers are linked when they have collaborated on some software code. This social network between software developers creates the backbone of a community where developers can access knowledge and experience of fellow developers. The time needed to resolve a software defect could be shortened by accessing developers who possess tacit knowledge about the software architecture (Kuk 2006) and have experience with debugging a particular module. Developers who are positioned in a network high in density (Coleman 1988) create a joint understanding of the software architecture and their relationships allow for the transfer of 'thick' information. An important motivation for developers to share knowledge with fellow developers is to build a reputation (Burt 2005; Hertel et al. 2003; Roberts et al. 2006; Shah 2006; Wasko and Faraj 2005). In dense networks, reputations of individuals are stable over time as the decay of relationships is low and the visibility of someone's actions are high as the information asymmetry concerning behavior is low (Coleman 1988; 1990). The stability of a dense network makes it worthwhile for a developer to make the investment of sharing time and knowledge with fellow developers. Dense networks facilitate the transfer of knowledge and build effective reputation mechanisms that are important to help a software developer in solving a software defect quicker. Hence, I hypothesize:

> **Hypothesis 3 (Dense Network):** The repair time of a software defect decreases when a software developer who is responsible for fixing the defect is embedded in a dense network.

## *How Community Members Shorten the Repair Time of a Bug*

Active community members are crucial in quickly resolving software defects according to open source advocates (Raymond 1999; Stallman 1992). At first, it seems counter-intuitive that a non-specialist, i.e. a community member, is capable in (co-) solving a software defect. While not all users will help fixing software defects, a small percentage of users will become bug reporters and do help. Two factors explain this type of behavior. First, a user benefits greatly by helping to solve a defect as that is the person who was using the functionality that was hiding the defect. It is in the best interest of the user to help solve the bug (von Hippel 1998). Second, open source software makes it possible to develop shared representation and understanding of the architecture (Raymond 1999) between bug reporters and developers. This is not the case with proprietary software where users do not have access to the source code. An understanding of the software architecture by community members helps in pinpointing the exact cause of the defect, and gives developers detailed information to replicate the defect. Third, short feedback loops between developers and bug reporters and short release cycles result in continuous improvements of the software. The visible progress of improvements motivates both developers and bug reporters to continue improving the software code.

The easiest way to become an active user is by filing a bug report which is the first step in solving a software defect. A bug report states what the user was trying to achieve, what actually happened and what should have happened. In addition, a bug report may contain details about the version of the software used, the operating system, and other technical details that help a developer in replicating the observed behavior.

> **Hypothesis 4 (Quality Report):** The repair time of a software defect decreases as the quality of the bug report submitted by a bug reporter increases

Community members who accumulate experience in writing bug reports will become better bug reporters over time. Experience in writing bug reports translates in reports that are more specific in stating the steps to replicate the anomaly and in the possible solutions of the anomaly. Task experience translates in improved performance as the bug reporters filing a bug report is more likely to have developed a personal relationship with the developers and can draw from previous experiences to know what kind of information developers are looking for. Hence, I hypothesize:

> **Hypothesis 5 (Bug Reporter Experience):** The repair time of a software defect decreases as the experience of the bug reporter increases.

Besides submitting bug reports, community members can help reduce the time needed to solve a defect by joining the discussion about the solution of a particular bug. Discussion between community members generates a diverse set of ideas about possible causes and solutions (Kuk 2006). Such a discussion can prevent the community from settling to quick on a solution and the discussion may help in mobilizing extra developers to solve the defect. A broad discussion about a particular defect makes the issue more visible and this increases the potential reputation benefits to fix the defect. The final hypothesis states:

> **Hypothesis 6 (Discussion Activity):** The repair time of a software defect decreases as the discussion activity about the resolution of the software defect increases.

## Research Setting

I chose the Firefox browser as my research setting for two reasons. First, the functionalities and expected outputs of a web browser are easier to understand by users compared with other software projects like operating systems (Linux), web servers (Apache), and compilers (GNU C). Second, Firefox meets the precondition of having a large active community that is using the browser. A large and active community, in relation to the size and complexity of the codebase, is a precondition to make the 'quality is community-based' methodology work (Raymond, 1999). These two reasons combined make the Firefox project a suitable research setting to test my hypotheses.

I collected data from the Firefox OSS project. Firefox is a popular browser used to surf the world-wide-web and developed under the auspices of Mozilla.org[3]. Firefox contains parts from the Mozilla source code, which is the open source descendant of the Netscape browser[4]. I collected data from the Bugzilla database and the Firefox Concurrent Versions System (CVS). Bugzilla is a tool used by the Firefox developers to track bugs and communicate about the possible resolution. In addition, Bugzilla acts as a knowledge repository for developers and bug reporters to store previous bugs and their solutions. A CVS system keeps track of all the individual files and changes to the files including a history of which developer changed what and when. I downloaded 25.249 Firefox related bugs for the period January 1999 – November 2008[5]. Of those, 7.128 bugs were fixed, the remainder of those bug reports are duplicates which will not be fixed (see The Life Cycle of a Bug for a more detailed explanation). Before explaining the data collection and discussing the variables, I first briefly discuss the life cycle of a bug.

### *The Life Cycle of a Bug*

When a user experiences an anomaly while using Firefox, for example, the browser does not respond as expected, certain functionalities do not work, or the browser crashes, s/he might file a bug report. I will refer to this person as the *bug reporter*. A new bug filed in Bugzilla is flagged as 'unconfirmed'. This unconfirmed bug is assigned to the developer who has ownership of the module in which the bug is presumably located and has the credentials to make changes to the source code. This decision is based on the initial bug report. Community members will try to replicate the bug. The person who tried to replicate the bug can choose from a number of different resolutions depending on the bug. The bug can be classified as 'duplicate' when the bug has been filed before. The bug can be classified as 'worksforme' meaning that nobody was able to replicate the bug. The bug can be classified as 'invalid'; the cause of the bug is not in the domain of Firefox but in other domains such as the operating system for example. The bug can be classified as 'wontfix'. This happens rarely, but for technical reasons including cross-browser compatibility, the bug does not get fixed. Finally, a bug can be classified as 'new'; these bugs are genuine and need to be fixed. Bugs that are not classified as 'new' are considered 'resolved'. A developer might take responsibility for fixing the bug and become the *bug owner* once someone replicates and confirms the bug as 'new'. Finally, when the bug owner fixes the bug and the solution is approved by the community then the bug's status becomes 'fixed'. Fixed bugs are verified by quality assurance: if a bug passes the quality tests then it is labeled as 'verified', if the bug does not pass quality assurance it is labeled as 'reopen'. Finally, 'verified' bugs are closed.

[3] Mozilla.org is the parent organization of the Firefox project. Mozilla.org provides, among other things, collaborative tools to develop Firefox such as the Bugzilla database and the Concurrent Versions System.

[4] Hammerly et al. (1999) discuss the motives behind the decision of Netscape to open source the source code of the Netscape browser.

[5] Mozilla.org released the first version of Firefox on September 23, 2002. Firefox contains source code from Mozilla and some bugs that affect Firefox originate from this source code. Hence, some bugs were filed before the Firefox project was started. Before settling on the name Firefox, the browser was first called Phoenix and later Mozilla Firebird. All these names refer to the same product.

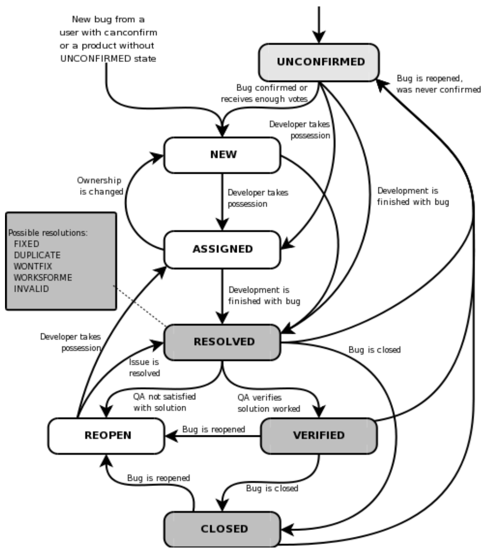
New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

**Figure 1. Life Cycle of a Firefox Bug**

The bug reporter and bug owner can be the same person but it is more common that these are different persons. Figure 1 summarizes the life cycle of a bug (Bugzilla.org 2008).

## *Data Collection*

I downloaded a detailed history for each bug. This history contains the date and time about each single change to the bug. The history of a bug includes updates in the status of a bug, who is responsible for fixing it, the discussion surrounding the solution, steps to reproduce the bug and the versions of Firefox that are affected. Second, I downloaded all Firefox source files including the history of each file. I retrieved the history files from the Firefox CVS server and these files contain detailed information about which developer added source code including date, time, and whether the source code fixed a specific bug.

## *Network Definition*

I use the CVS history log file to construct the developer network. Each time a developer commits source code to the CVS trunk information about the commit is added to the history log file. This information contains date and time of the commit, name of the developer who made the commit, the bug id from the Bugzilla database when the commit fixes a software defect, the names of the developers who are responsible for quality control and the acknowledgements to developers who helped coding the patch. Acknowledging other developers contribution is a very strong norm in open source communities (Raymond 1999; Stewart and Gosain 2006) and people who deviate from this norm lose respect in the community and will socially isolate themselves. Deviance of this norm, or "surreptitiously filing someone's name off a project is, in cultural context, one of the ultimate crimes. Doing this steals the victim's gift to be presented as the thief's own." according to Raymond (1999). Acknowledgements built reputations and act as a signaling device (Lerner and Tirole 2002). Hence, the history log files give detailed and accurate information about the interpersonal relationships between developers. The developer network is undirected, as I cannot make assumptions about who initiated a relationship.

## *Model Specification & Dependent Variable*

I use as my dependent variable the time needed to fix a bug. I define the repair time of a software defect as the number of days that elapse between the day that bug was reported in the Bugzilla database and the date the bug was labeled 'fixed' in the Bugzilla database. I code my dependent variable a '0' as long as the bug is not labeled as 'fixed' and a '1' when the bug is labeled as 'fixed'. I will use survival analysis to model the process of fixing a bug. Survival analysis is an appropriate method as each bug, at the initial reporting, is at risk of being solved. In particular, I use Cox regression to test which factors lengthen and shorten the time needed to fix a bug. Cox regression is a parametric regression model that does not make any assumptions about the shape of the hazard over time nor is it estimated but the hazard is held constant over observations. Predictor variables are multiplicative and move the hazard along the y-axis but do not change the shape of the hazard. To ease the interpretation, I report coefficient estimates instead of the hazard ratios[6]. Positive coefficients indicate that a factor shortens the time needed to solve a software defect while negative coefficients indicate that a factor lengthens the time needed to solve a software defect.

## *Independent & Control Variables*

**Table 1. Independent Variables**

| | |
|---|---|
| *Community (Hypothesis 1-2)* | |
| **Bug reporters** | For each month during the 10-year observation period, I calculate the number of active Firefox community members by counting the unique number of email-addresses. This is consistent with the previous operationalization by Butler (2001). |
| **Active Developers** | For each month during the 10-year observation period, I calculate the number of active Firefox developers by parsing the commit history files and identify developers using their email address. I built a large dictionary that maps different email addresses to a single developer as some developers change their email address during the observation window. |
| *Network of software developer (Hypothesis 3)* | |
| **Dense Network** | I measure the extent to which a developer occupies a dense network position by calculating the ego density of the network of an individual developer. Ego density is defined as t / N*(N-1) where t is the number of relationships between the direct colleagues of the software developer and N is the number of direct colleagues of the software developer (Coleman, 1988). |
| *Quality of bug report (Hypothesis 4-6)* | |
| **Quality** | I measure the quality of a bug report using the five items described below. I sum these five items to create an overall 'quality' variable that ranges between 0 and 5. '0' indicates that the bug report does not contain any of items while '5' indicates that a bug report contains all items. |
| Item 1: **Steps** | A dummy variable coded '1' when the bug report contains explicit steps to reproduce the bug or '0' otherwise. Explicit steps reduce the time to replicate a bug as they instruct other people how to replicate the defect. |
| Item 2: **Screenshot** | A dummy variable coded '1' when the bug report contains a link to a screenshot or '0' otherwise. Screenshots can help explain the nature of the problem or suggest the expected result. |
| Item 3: **Stacktrace** | A dummy variable coded '1' when the bug report contains a stacktrace and '0' otherwise. A stacktrace is a dump of the memory just before an application crashes and gives developers valuable debug information. |
| Item 4: **Attachment** | A dummy variable coded '1' when the bug report contains an attachment to illustrate the problem and '0' otherwise. |
| Item 5: **Version Information** | A dummy variable coded '1' when the bug report contains specific information regarding the browser and operating system version or '0' otherwise. |

Other factors, besides the hypothesized effects of the community, user contributions, and social networks of the developers may affect the time needed to fix a bug. I include control variables to rule out alternative explanations. Table 2 gives an overview.

[6] These coefficient estimates are interpreted similarly as coefficients from ordinary least squares regression. The hazard ratios can be obtained by taking the exponent of the coefficient.

**Table 2. Control Variables**

| | |
|---|---|
| *Community Related Control Variables* | |
| **Community Cohesion** | Networks that are more cohesive have fewer potential bottlenecks that could stall information transfer. I add a network density variable, calculated as t / (N*(N-1)) where t is the number of observed relationships and N is the number of developers. |
| **General Workload** | I control for general workload by counting the number of 'open' bugs. An increasing workload will diminish the likelihood that a particular defect gets fixed. |
| **Year Dummies** | I include 10-year dummies (1998 - 2008) to control for temporal changes in the Firefox community. These year dummies should capture effects such as the overall popularity of the Firefox browser and the increasing complexity of the software due to the addition of new features. |
| *Software Defect Related Control Variables* | |
| **Length** | This variable counts the number of words in the initial bug report. More detailed bug reports make it easier to replicate the bug and take necessary steps. |
| **Bug Comments** | This variable counts the total number of comments made by developers and users during the life cycle of the bug. Increased communication between developers is likely to shorten the time to fix a bug. |
| **Title Length** | This variable counts the number of words in the title of the bug report. Titles that are more descriptive help in assigning the bug to the appropriate developer. |
| **Votes** | This variable counts the total number of votes. Developers and users can vote whether a particular bug should be fixed. Bugs with more votes are more likely to be fixed because the Firefox community deems them more important. |
| **Platform** | A dummy variable that indicates which operating system Firefox is affected. Some bugs only appear on certain operating systems while other bugs are platform independent. |
| **Milestone** | A dummy variable coded a '1' if the bug has been designated to be solved before a certain milestone and '0' otherwise. |
| **Regression** | A dummy variable coded a '1' if the bug has been designated as a regression. A regression, in software development terms, refers to a change in the software code that causes a previously fixed bug to reappear. The source coded has regressed to a previous non-working state. |
| **Severity** | A dummy variable that can have the following values: 'blocker', 'critical ', 'major ', 'normal', 'minor', enhancement, and 'trivial'. The severity of a bug can change throughout the life cycle of the bug. Severe bugs are more likely to be fixed sooner than later. |
| **Priority** | A dummy variable coded '1' to '5'. When this variable is coded '1' it refers to high priority because the bug causes reproducible crashes, or creates a serious security issue or creates a serious problem on an important web site. A '5' indicates low priority because the bug is a minor issue, cosmetic problem or creates a very rare problem. The priority of a bug can change throughout the life cycle of the bug. |
| **Bug complexity** | When the resolution of one bug depends or blocks the resolution of another bug then those two bugs are connected. I retrieved this information from the Bugzilla database. I constructed day-to-day bug dependency networks, in total 2.380 networks, and measure the bug complexity for each bug by counting the number of bugs that are blocked by the focal bug.[7] |

[7] For days in which the bug-dependency network did not change, I did not create a new network but instead I used the network of the previous day.

| *Developer Related Control Variables* | |
|---|---|
| **Developer Workload** | A variable counting the number of unresolved bugs assigned to the developer at any given moment in time. A higher workload reduces the likelihood that any bug gets fixed. I take the natural logarithm to reduce the skewness of this variable. |
| **Employer** | A dummy variable coded '1' if Mozilla.org employs the developer. The email address that the developer uses to commit source code is used to determine whether the developer is a volunteer or is paid by Mozilla.org. Developers paid to develop for the Firefox project have more resources to spend on fixing a bug compared with volunteers. |
| **Developer Experience** | Developer experience will most certainly affect the time needed to repair a software defect. More experienced developers will require less time. Hence, to control for this alternative explanation, I use two variables to measure experience. The first experience variable is a running clock starting when a developer made her first commit to the software trunk. The second experience variable is a count of the total number of commits made. I prefer to control for developer experience than for developer productivity. Measuring developer productivity by looking at the output as measured in source lines of code (SLOC) (Chidamber et al. 1998) can be misleading because experienced programmers use fewer lines of code to implement a certain functionality compared with inexperienced programmers who implement the same functionality (von Hippel and von Krogh 2003). Hence, I prefer to control for developer experience, as this should capture developer productivity effects as well. |
| **Size of developer's network** | A variable counting the total number of relationships the developer has, based on the source code commits made in the last month. I use degree to calculate this variable. |

## Results

Table 1 presents the main statistics and correlations of our variables. Scores for the variance inflation factors (VIF) are well below the rule of thumb of 10 (2.42 on average) (Baum 2006) and the low bivariate correlations suggest that multicollinearity is not a salient issue. Table 2 presents five Cox regression models. Model 1 is a baseline model only containing control variables. Models 2 to 4 test the main effects of the community (Model 2), the social network of developers (Model 3) and user contributions (Model 4). Finally, Model 5 is the integrated model showing all variables. I use Model 5 to discuss the results, as the findings are consistent across the different models.

Model 5 gives strong support for the 'bug reporter (H1) hypothesis. The time needed to resolve a software defect decreases as more people become bug reporters in the Firefox community. In contrast, there is no support for the 'active developer' hypothesis (H2). The time needed to solve a bug increases as more developers join the Firefox community. This could suggest that there are potential bottlenecks in reviewing fixes to the codebase of Firefox or that it takes significant amount of time of the senior developers in embedding the new developers in the community. Time they could have spend solving bugs is spend on maintaining community cohesion. This alternative explanation of the 'active developer' hypothesis is supported by the 'dense network' hypothesis (H3). Strongly embedded software developers in a dense network are better equipped in tapping into the advice and experience of fellow developers. Fellow developers are less uncertain about the quality of the work as the reputational mechanism of a dense network reduces information asymmetries between developers. Thus, I find strong support that strongly embedded developers require less time to solve a defect.

The final three hypotheses focus on the impact of user contributions. The 'quality report' hypothesis (H4) predicted that more detailed bug reports would require less time to fix as developers who have valuable information to pinpoint the cause of the anomaly. I do not find support for this prediction. On the contrary, detailed bug reports require significant more time to fix compared with less detailed bug reports. The 'bug reporter experience' hypothesis (H5) is not supported by the models. Community members do not seem to become more experienced in writing bug reports. A possible explanation could be that the churn of members entering and leaving the Firefox community is too high and hence community members do not go through the learning curve of writing high quality bug reports. A second alternative explanation is that community members and software developers do not develop a shared understanding of the software architecture as suggested by Raymond (1999). The different models support the discussion activity hypothesis (H6). Intensively discussed bugs are more quickly resolved compared with bugs that are less discussed.

**Table 3. Descriptives & Correlations**

| | Variables | Mean | S.D. | Min | Max | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | Bug fixed (dummy) | 0.14 | 0.35 | 0 | 1 | 1 | | | | | | | | | | | | | | | | | | |
| **2** | Number of words of initial bug report | 118.27 | 180.42 | 1 | 9533 | -0.133 | 1 | | | | | | | | | | | | | | | | | |
| **3** | Number of words of bug report title | 9.52 | 4.49 | 1 | 49 | -0.048 | 0.096 | 1 | | | | | | | | | | | | | | | | |
| **4** | Number of votes bug received | 0.42 | 3.15 | 0 | 256 | 0.079 | -0.005 | 0.017 | 1 | | | | | | | | | | | | | | | |
| **5** | Bug has been flagged as regression (dummy) | 0.05 | 0.22 | 0 | 1 | 0.255 | -0.039 | 0.02 | 0.038 | 1 | | | | | | | | | | | | | | |
| **6** | Bug complexity | 0.03 | 0.15 | 0 | 1 | 0.222 | -0.029 | 0.008 | 0.137 | 0.13 | 1 | | | | | | | | | | | | | |
| **7** | Developer is employed by Mozilla.org (dummy) | 0.15 | 0.35 | 0 | 1 | 0.292 | -0.041 | -0.001 | 0.026 | 0.057 | 0.049 | 1 | | | | | | | | | | | | |
| **8** | Network size of individual developer | 6.4 | 16.36 | 0 | 155 | 0.312 | -0.051 | -0.014 | 0.023 | 0.124 | 0.064 | 0.05 | 1 | | | | | | | | | | | |
| **9** | Developer workload (log) | 4.94 | 2.06 | 0 | 8.41 | 0.016 | -0.005 | 0.048 | 0.013 | 0.049 | 0.018 | 0.06 | 0.242 | 1 | | | | | | | | | | |
| **10** | Developer experience (years) | 1.42 | 1.11 | 0 | 7.23 | 0.3 | -0.041 | -0.007 | 0.064 | 0.138 | 0.122 | 0.132 | 0.288 | 0.476 | 1 | | | | | | | | | |
| **11** | Developer experience (commits / 100) | 6.25 | 9.06 | 0.01 | 61.74 | 0.156 | -0.026 | 0.037 | 0.027 | 0.09 | 0.041 | 0.039 | 0.337 | 0.687 | 0.42 | 1 | | | | | | | | |
| **12** | Number of bugs waiting for solution (log) | 9.03 | 0.81 | 3.37 | 9.83 | 0.253 | -0.01 | 0.056 | 0.062 | 0.105 | 0.093 | 0.172 | 0.068 | 0.208 | 0.43 | 0.248 | 1 | | | | | | | |
| **13** | Community cohesion | 1.15 | 1.39 | 0 | 4.27 | 0.187 | -0.015 | 0.002 | 0.021 | 0.056 | 0.026 | 0.073 | 0.331 | 0.343 | 0.298 | 0.321 | 0.104 | 1 | | | | | | |
| **14** | Number of active members of Firefox community (*1000) | 18.2 | 12.54 | 0.99 | 50.52 | 0.493 | -0.074 | 0.016 | 0.094 | 0.18 | 0.183 | 0.184 | 0.261 | 0.22 | 0.577 | 0.317 | 0.755 | 0.138 | 1 | | | | | |
| **15** | Number of active developers of Firefox | 92.03 | 115.87 | 0 | 468 | 0.426 | -0.066 | -0.014 | 0.03 | 0.139 | 0.081 | 0.145 | 0.535 | 0.324 | 0.429 | 0.412 | 0.268 | 0.724 | 0.463 | 1 | | | | |
| **16** | Network closure of developer | 9.22 | 18.15 | 0 | 100 | 0.33 | -0.055 | -0.033 | 0.041 | 0.1 | 0.058 | 0.146 | 0.204 | 0.099 | 0.224 | 0.138 | 0.148 | 0.533 | 0.202 | 0.558 | 1 | | | |
| **17** | Quality of bug report | 1.78 | 1.1 | 0 | 5 | -0.49 | 0.226 | 0.153 | -0.016 | -0.128 | -0.119 | -0.192 | -0.232 | -0.06 | -0.238 | -0.12 | -0.129 | -0.135 | -0.38 | -0.325 | -0.25 | 1 | | |
| **18** | Bug reporter experience (years) | 0.56 | 1.01 | 0 | 6.46 | 0.507 | -0.128 | -0.06 | 0.02 | 0.228 | 0.16 | 0.168 | 0.237 | 0.092 | 0.419 | 0.175 | 0.334 | 0.103 | 0.572 | 0.351 | 0.24 | -0.496 | 1 | |
| **19** | Discussion activity | 7.94 | 12.2 | 1 | 517 | 0.359 | -0.029 | 0.017 | 0.533 | 0.14 | 0.286 | 0.067 | 0.127 | 0.024 | 0.169 | 0.077 | 0.147 | 0.071 | 0.287 | 0.162 | 0.113 | -0.145 | 0.18 | 1 |
| | 25.278 Observations | | | | | | | | | | | | | | | | | | | | | | | |
| | Correlations > \|0.03\| are significant at 5% level | | | | | | | | | | | | | | | | | | | | | | | |

**Table 4. Results Cox Regression**

| | Hypothesis | Control Model | | Community Model | | Developer Network Model | | User Contributions Model | | Complete Model | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Variables* | | b | se | b | se | b | se | b | se | b | se |
| Bug has been flagged for milestone (dummy) | | 0.439 | 0.033 *** | 0.427 | 0.033 *** | 0.435 | 0.033 *** | 0.304 | 0.033 *** | 0.282 | 0.033 *** |
| Number of words of initial bug report | | -0.002 | 0.000 *** | -0.002 | 0.000 *** | -0.002 | 0.000 *** | -0.001 | 0.000 ** | -0.001 | 0.000 ** |
| Number of words of bug report title | | -0.018 | 0.003 *** | -0.018 | 0.003 *** | -0.018 | 0.003 *** | -0.015 | 0.003 *** | -0.014 | 0.003 *** |
| Number of votes bug received | | 0.021 | 0.003 *** | 0.020 | 0.003 *** | 0.021 | 0.003 *** | -0.002 | 0.004 | -0.007 | 0.004 |
| Bug has been flagged as regression (dummy) | | 0.099 | 0.038 * | 0.098 | 0.038 * | 0.089 | 0.039 * | 0.125 | 0.038 ** | 0.106 | 0.038 ** |
| Bug complexity | | -0.258 | 0.052 *** | -0.285 | 0.052 *** | -0.254 | 0.052 *** | -0.381 | 0.053 *** | -0.416 | 0.053 *** |
| Developer is employed by Mozilla.org (dummy) | | -0.127 | 0.027 *** | -0.101 | 0.027 *** | -0.132 | 0.027 *** | -0.176 | 0.027 *** | -0.148 | 0.027 *** |
| Network size of individual developer | | 0.003 | 0.001 *** | 0.005 | 0.001 *** | 0.003 | 0.001 *** | 0.003 | 0.001 *** | 0.005 | 0.001 *** |
| Developer workload (log) | | -0.228 | 0.010 *** | -0.234 | 0.010 *** | -0.227 | 0.010 *** | -0.231 | 0.010 *** | -0.234 | 0.010 *** |
| Developer experience (years) | | 0.104 | 0.012 *** | 0.097 | 0.012 *** | 0.099 | 0.012 *** | 0.109 | 0.012 *** | 0.097 | 0.013 *** |
| Developer experience (commits / 100) | | 0.009 | 0.002 *** | 0.011 | 0.002 *** | 0.009 | 0.002 *** | 0.012 | 0.002 *** | 0.014 | 0.002 *** |
| Number of bugs waiting for solution (log) | | 0.365 | 0.074 *** | 0.073 | 0.079 | 0.362 | 0.074 *** | 0.306 | 0.074 *** | 0.029 | 0.078 |
| Community cohesion | | 0.315 | 0.012 *** | 0.372 | 0.015 *** | 0.293 | 0.013 *** | 0.290 | 0.012 *** | 0.319 | 0.015 *** |
| Number of active members of Firefox community | H1(+) | | | 0.043 | 0.006 *** | | | | | 0.039 | 0.006 *** |
| Number of active developers of Firefox | H2(+) | | | -0.001 | 0.000 *** | | | | | -0.001 | 0.000 *** |
| Network closure of developer | H3(+) | | | | | 0.002 | 0.001 ** | | | 0.005 | 0.001 *** |
| Quality of bug report | H4(+) | | | | | | | -0.384 | 0.015 *** | -0.380 | 0.015 *** |
| Bug reporter experience (years) | H5(+) | | | | | | | 0.003 | 0.011 | -0.003 | 0.011 |
| Discussion activity | H6(+) | | | | | | | 0.010 | 0.001 *** | 0.010 | 0.001 *** |
| *Dummy control variables included* | | *Yes* | | *Yes* | | *Yes* | | *Yes* | | *Yes* | |
| Number of observations | | 25.278 | | 25.278 | | 25.278 | | 25.278 | | 25.278 | |
| Number of bug fixes | | 7.128 | | 7.128 | | 7.128 | | 7.128 | | 7.128 | |
| Log likelihood | | -47297.92 | | -47236.72 | | -47292.68 | | -46864.87 | | -46788.69 | |
| 2LL | | | | 122.4 *** | | 10.48 ** | | 866.1 *** | | 1018.46 *** | |

* p<0.05, ** p<0.01, *** p<0.001

The dummy control variables are not of theoretical interest and space limitations do not allow me to present these variables.

Overall, these findings suggest that an open source community plays in important role in reducing the time required to solve a bug. Larger communities require less time to fix a bug but there are challenges in incorporating new developers. Dense networks are important for the transfer of tacit knowledge and the building of reputations for software developers. The role of user contributions in fixing software defects is more nuanced: I did not find support for the quality of the bug report or the experience of the reporter in shortening the time required to fix a bug. I did find support that active discussions are important in finding a solution for a software anomaly.

### *Alternative Specifications*

I did three robustness checks. First, the finding that more detailed bug reports need more time to solve is surprising. I tested a model using the individual items for the quality of a bug report to check the construct validity. The model with the five separate items supports the findings in this paper. Second, I used the network constraint measure of Burt as an alternative operationalization for the dense network measure. High network constraint indicates the presence of ties between fellow developers. This model confirmed the findings in this paper. Finally, I used robust standard errors to calculate more conservative *t* values; this did not alter my findings. Space limitations prevent me from showing these models in this paper.

## Conclusions

I started this paper with the observation that a central tenet of the open source philosophy is that software developed using the open source method leads to software quality that is at least comparable with commercial products or, as some would argue, superior to commercial products. Software defects are quickly detected and resolved by exposing the source code to a potential large audience of developers and users according to the open source software methodology (Raymond, 1999). Open source projects rely in large parts on their communities instead of relying only on formal methodologies to improve the quality of software. I adopted a micro-level perspective by zooming in on individual software defects and I studied how an open source community helps in fixing software defects. Before drawing conclusions, I briefly discuss limitations, practical implications, and future research.

### *Limitations*

A valid concern about the estimates of the presented models is that developers self-select which bugs they fix. Self-selection poses a problem in establishing causality. Ideally, one would include a developer quality or productivity control variable to neutralize self-selection, especially since the variance in productivity between developers is significant (Brooks 1975; Raymond 1999). However, developer quality is hard to observe and measure. I included developer experience as alternative control but I cannot completely rule out a self-selection effect.

### *Practical Implications*

This study suggests that a growing and vibrant community of bug reporters and developers is essential in improving the quality of software of an open source project. Two practical implication of this finding come to mind. First, developers of open source projects could lower barriers to entry for end-users by teaching bug writing skills and educating them about the type of information developers need to replicate an anomaly. Second, the recruitment of new developers and bug reporters (Oh and Jeon 2007; Roberts et al. 2006) and the integration of these new members in the community is essential to the long-term viability of an open source project.

### *Future Research*

Open source communities contain detailed longitudinal data on collaboration, developer productivity, and network evolution which is freely available. As such, this presents a fruitful empirical context for scholars with a diverse range of interests. I will highlight some promising avenues for future research. Measuring the productivity of a developer has proven to be very difficult (Krishnan et al. 2000) due to large variations between developers in competence (Weinberg 1971), the lack of access to the source code and the difficulty in tying the efforts of a single developer to a particular piece of software code (Lerner and Tirole 2002). Open source software provides a research setting were the contributions of individuals can be linked to software functionality in combination with the availability of source code and detailed activity information makes it possible to start addressing the topic of developer productivity again.

Concluding, I find strong evidence that an open source community plays a pivotal role in reducing the time needed to fix software bugs. My research suggests that an open source community helps in three distinct ways. First, the size of a community matters. An open source community that grows in size has more people who actively help in finding the cause of a defect and debating about the possible solution. This finding reiterates the observation made by Raymond (1999) who stated "given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone". Thus, I do find strong support for Linus Law that "given enough eyeballs, all bugs are shallow". Second, how a software developer is embedded in the larger structure of the community is consequential for the time needed to fix a bug. Developers who have a dense network can tap the knowledge and experiences of other developers and fellow developers benefit from contributing by building a stable reputation over time. This shortens the time needed to resolve a defect. Finally, user contributions are important in helping developers to pinpoint the cause of a defect. However, the findings concerning the user contributions demonstrated a more complicated picture. High quality bug reports filed by community members did not shorten the time needed to resolve a defect; on the contrary, high quality bug reports lengthened the time needed to fix a bug. Two alternative explanations of this finding are possible. First, the cognitive distance between bug reporter and developer is too large. Community members try to give as much relevant information but do not know *a priori* what a developer needs to know. Second, the detailed information could put a developer on the wrong track. The developer uses the information from the bug report to pinpoint the cause but the bug report contains too much noise. The findings of this research suggest that the community of members and developers is important for making a bug 'shallow' and the bazaar metaphor might still be quite appropriate.

## Acknowledgements

This research is made possible by the Rubicon grant from The Netherlands Organisation for Scientific Research (NWO). I am thankful for feedback from Joel Baum, Hans Frankort, Eric van Heck, Alex Oettl, Tim Rowley, Peter Vervest, the Mozilla Toronto office, in particular Mike Beltzner and Mark Surman, and the insightful comments from the Associate Editor and the two anonymous reviewers of the International Conference on Information Systems. All errors and omissions remain my own.

## References

A.P. 2006. "Software Delay Said to Cost Irs $318 Million in Overpaid Refunds." Retrieved April 28, 2009, from http://www.washingtonpost.com/wp-dyn/content/article/2006/09/01/AR2006090101507.html

Adams, D.A., Nelson, R.R., and Todd, P.A. 1992. "Perceived Usefulness, Ease of Use, and Usage of Information Technology - a Replication," *MIS Quarterly* (16:2), pp 227-247.

Bagozzi, R.P., and Dholakia, U.M. 2006. "Open Source Software User Communities: A Study of Participation in Linux User Groups," *Management Science* (52:7), Jul, pp 1099-1115.

Banker, R.D., Datar, S.M., and Kemerer, C.F. 1991. "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects," *Management Science* (37:1), Jan, pp 1-18.

Banker, R.D., and Kauffman, R.J. 2004. "The Evolution of Research on Information Systems: A Fiftieth-Year Survey of the Literature in Management Science," *Management Science* (50:3), Mar, pp 281-298.

Baum, C.F. 2006. *An Introduction to Modern Econometrics Using Stata*. College Station, TX: Stata Press Publication.

Brooks, F.P. 1975. *The Mythical Man-Month - Essays on Software Engineering*. Addison Wesley.

Bugzilla.org. 2008. "The Bugzilla Guide - 3.2 Release." from http://www.bugzilla.org/docs/3.2/en/html/lifecycle.html

Burt, R.S. 1992. *Structural Holes - the Social Structure of Competition*. Cambridge, MA: Harvard University Press.

Burt, R.S. 2005. *Brokerage and Closure: An Introduction to Social Capital*. Oxford, UK: Oxford University Press.

Butler, B.S. 2001. "Membership Size, Communication Activity, and Sustainability: A Resource-Based Model of Online Social Structures," *Information Systems Research* (12:4), pp 346-362.

Butler, B.S., and Gray, P.H. 2006. "Reliability, Mindfulness, and Information Systems," *MIS Quarterly* (30:2), Jun, pp 211-224.

Chidamber, S.R., Darcy, D.P., and Kemerer, C.F. 1998. "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Transactions on Software Engineering* (24:8), Aug, pp 629-639.

Coleman, J.S. 1988. "Social Capital in the Creation of Human-Capital," *American Journal of Sociology* (94), pp S95-S120.

Coleman, J.S. 1990. *Foundations of Social Theory*. Cambridge, MA: Harvard University Press.

Crosby, P. 1979. *Quality Is Free*. New York, NY: McGraw-Hill.

Cusumano, M.A., and Kemerer, C.F. 1990. "A Quantitative-Analysis of United-States and Japanese Practice and Performance in Software-Development," *Management Science* (36:11), Nov, pp 1384-1406.

Dinh-Trong, T., and Bieman, J.M. 2004. "Open Source Software Development: A Case Study of FreeBSD," *10th International Symposium on Software Metrics*, Chicago, IL: IEEE Computer Soc, pp. 96-105.

Eisenstadt, M. 1997. "My Hairiest Bug War Stories," *Communications of the ACM* (40:4), pp 30-37.

Fitzgerald, B. 2006. "The Transformation of Open Source Software," *MIS Quarterly* (30:3), Sep, pp 587-598.

Hammerly, J., Paquin, T., and Walton, S. 1999. "Freeing the Source: The Story of Mozilla," in: *Open Sources: Voices from the Open Source Revolution,* C. DiBona, S. Ockman and M. Stone (eds.). Sebastopol, CA.: O'Reilly, pp. 197-206.

Harter, D.E., Krishnan, M.S., and Slaughter, S.A. 2000. "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science* (46:4), pp 451-466.

Henderson, J.C., and Lee, S. 1992. "Managing I/S Design Teams - a Control Theories Perspective," *Management Science* (38:6), Jun, pp 757-777.

Hertel, G., Niedner, S., and Herrmann, S. 2003. "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* (32:7), Jul, pp 1159-1177.

Hoopes, D.G., and Postrel, S. 1999. "Shared Knowledge, "Glitches," And Product Development Performance," *Strategic Management Journal* (20:9), Sep, pp 837-865.

ISO. 1996. "ISO 9126 - Software Quality Characteristics."

Kidwell, P.A. 1998. "Stalking the Elusive Computer Bug," *IEEE Annals of the History of Computing* (20:4), pp 5-9.

Krishnan, M.S., Kriebel, C.H., Kekre, S., and Mukhopadhyay, T. 2000. "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science* (46:6), pp 745-759.

Kuk, G. 2006. "Strategic Interaction and Knowledge Sharing in the Kde Developer Mailing List," *Management Science* (52:7), Jul, pp 1031-1042.

Lakhani, K.R., and von Hippel, E. 2003. "How Open Source Software Works: "Free" User-to-User Assistance," *Research Policy* (32:6), pp 923-943.

Lerner, J., and Tirole, J. 2002. "Some Simple Economics of Open Source," *Journal of Industrial Economics* (50:2), Jun, pp 197-234.

MacCormack, A., Rusnak, J., and Baldwin, C.Y. 2006. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science* (52:7), Jul, pp 1015-1030.

Mockus, A., Fielding, R.T., and Herbsleb, J.D. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11:3), Jul, pp 309-346.

Oh, W., and Jeon, S. 2007. "Membership Herding and Network Stability in the Open Source Community: The Ising Perspective," *Management Science* (53:7), Jul, pp 1086-1101.

Olivera, F., Goodman, P.S., and Tan, S.S.L. 2008. "Contribution Behaviors in Distributed Environments," *MIS Quarterly* (32:1), Mar, pp 23-42.

Ravichandran, T., and Rai, A. 2000. "Quality Management in Systems Development: An Organizational System Perspective," *MIS Quarterly* (24:3), Sep, pp 381-415.

Raymond, E.S. 1999. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* Sebastopol, CA: O'Reilly & Associates, Inc.

Roberts, J.A., Hann, I.H., and Slaughter, S.A. 2006. "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management Science* (52:7), Jul, pp 984-999.

Shah, S.K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52:7), pp 1000-1014.

Slaughter, S.A., Harter, D.E., and Krishnan, M.S. 1998. "Evaluating the Cost of Software Quality," *Communications of the ACM* (41:8), Aug, pp 67-73.

Stallman, R. 1992. "Why Software Should Be Free." from http://www.gnu.org/philosophy/shouldbefree.html

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. 2006. "Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects," *Information Systems Research* (17:2), pp 126-144.

Stewart, K.J., and Gosain, S. 2006. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *MIS Quarterly* (30:2), pp 291-314.

Venkatesh, V., Morris, M.G., Davis, G.B., and Davis, F.D. 2003. "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly* (27:3), pp 425-478.

Vixie, P. 1999. "Software Engineering," in: *Open Sources: Voices from the Open Source Revolution,* C. DiBona, S. Ockman and M. Stone (eds.). Sebastopol, CA: O'Reilly.

von Hippel, E. 1998. "Economics of Product Development by Users: The Impact Of "Sticky" Local Information," *Management Science* (44:5), pp 629-644.

von Hippel, E. 2007. "Horizontal Innovation Networks - by and for Users," *Industrial and Corporate Change* (16:2), pp 293-315.

von Hippel, E., and von Krogh, G. 2003. "Open Source Software and The "Private-Collective" Innovation Model: Issues for Organization Science," *Organization Science* (14:2), pp 209-223.

von Krogh, G., Spaeth, S., and Lakhani, K.R. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy* (32:7), Jul, pp 1217-1241.

Wasko, M.M., and Faraj, S. 2005. "Why Should I Share? Examining Social Capital and Knowledge Contribution in Electronic Networks of Practice," *MIS Quarterly* (29:1), Mar, pp 35-57.

Weinberg, G.M. 1971. *The Psychology of Computer Programming*. New York, NY: Dorset House.

Wellman, B., Salaff, J., Dimitrova, D., Garton, L., Gulia, M., and Haythornthwaite, C. 1996. "Computer Networks as Social Networks: Collaborative Work, Telework, and Virtual Community," *Annual Review of Sociology* (22), pp 213-238.