Impact of switching bug trackers: a case study on a medium-sized open source project

Théo Zimmermann, Annalí Casanueva Artís

Impact of switching bug trackers: a case study on a medium-sized open source project

Théo Zimmermann
ΠR^2, Inria and IRIF, Paris-Diderot University
Paris, France
Email: theo@irif.fr

Annalí Casanueva Artís
Paris School of Economics
Paris, France

***Abstract*—For most software projects, the bug tracker is an essential tool. In open source development, this tool plays an even more central role as it is generally open to all users, who are encouraged to test the software and report bugs. Previous studies have highlighted the act of reporting a bug as a first step leading a user to become an active contributor.**

The impact of the bug reporting environment on the bug tracking activity is difficult to assess because of the lack of comparison points. In this paper, we take advantage of the switch, from Bugzilla to GitHub, of the bug tracker of Coq, a medium-sized open source project, to evaluate the impact that such a change can have.

We first report on the switch itself, in particular the migration of 4900 preexisting bug reports using the GitHub issue import API. Then we analyze data from before and after the switch using a Regression on Discontinuity analysis, a novel methodology in the context of empirical software engineering. We show that the switch induces an increase in bug reporting, particularly from principal developers themselves, and more generally an increased engagement with the bug tracking platform, with more comments by developers and also more external commentators.

***Index Terms*—bug tracker, switch, migration, bug report, issue, GitHub, Bugzilla, open source, data mining, Regression on Discontinuity**

I. Introduction

Bug reporting is an essential part of software development. In the context of an open source project, the bug reporting and fixing process is generally done on a public bug tracking platform, and, in the absence of paid testers, it depends a lot on the good will of independent users. Therefore, it involves a strong social component.

However, having an open bug tracking system is not enough to get the bugs reported. In addition to the software having enough users, the process of bug reporting must be easy and appealing. Therefore, creating an environment that eases bug reporting and makes it more appealing may lead to an increase in bug reporting activity (from both developers and independent users), which is well associated with an increase in software quality. First, if the software has not changed, receiving more bug reports indicates that more defects have been found, and not a sudden apparition of defects (Linus's Law "given enough eyeballs, all bugs are shallow" [1], [2]). Moreover, assuming that developers are equipped to cope with the increased number of incoming reports [3], [4], even duplicate reports are not necessarily harmful [5]. Second, more activity on the bug tracker can also mean users are helping to reproduce bugs, produce traces, etc. and thus are working with the developers to get the bugs fixed [6]. More generally, reporting new bugs and discussing existing bug reports has been shown to be an important step on the path to becoming an active contributor of an open source project [7], [8], [9].

Despite the importance of the bug tracking environment that we have just highlighted, the impact of a change in this environment has rarely been studied. It is generally difficult to compare the number of bug reports in different environments because it is likely to depend not just on the reporting environment but also on each particular software and its development dynamics, which would make such an analysis not compelling. To do a valid comparison, one would ideally need to compare two bug reporting environments for the same software, used by the same users, to report the same bugs, in the same period. This is of course impossible. A context approaching this ideal scenario would be to compare the number of bug reports of a particular software before and after the switch to a new environment. Yet, substantial changes in the environment of bug reporting (e.g. changes of bug tracker platform) are rare because once a bug tracking platform is in place, any change comes with a significant cost.

In this paper, we study the case of the Coq proof assistant, a medium-sized open source software project. We use the switch, from Bugzilla to GitHub, of the project's bug tracker, to analyze how a change in the bug reporting environment affects the number of bugs, reporters, bug comments and commentators. Using a Regression on Discontinuity design (RDD) [10], [11], [12], [13], we find strong evidence that the change of platform increased the number of bug reports and comments by developers, and increased the number of distinct commentators each week among non-developers. Our results suggest that switching from Bugzilla to GitHub made the bug reporting process easier and more appealing but also opened the software development to more external contributors and stimulated discussion around bugs (notably between developers and non-developers).

The contributions of this paper are practical, empirical, and methodological. First, we improved an existing bug tracker migration tool to handle thousand of bug reports while preserving meta-data, and we helped the Coq development team migrate their bug tracker from Bugzilla to GitHub. Second, we analyze the causal impact of this switch on the bug tracking activity. Third, we introduce to the field of software engineering a state-of-the-art method for quantitative public policy evaluation:

Regression in Discontinuity Design. In particular, we explain this method to give the basic intuition behind; we point the reader to the both introductory and state-of-the-art literature and we illustrate one application.

The rest of this paper is organized as follows: In Sec. II, we discuss related work. In Sec. III, we present the Coq project and the context of the switch. In Sec. IV, we formulate hypotheses on the impact of the switch. In Sec. V, we explain the migration process. In Sec. VI, we describe the data collection and pre-processing method, and we list the variables of interest. In Sec. VII, we give a preliminary view of the bug tracking activity. In Sec. VIII, we explain the Regression on Discontinuity methodology, of which we present the results in Sec. IX. We interpret the results and relate them to our hypotheses in Sec. X. We discuss the threats to validity and present robustness checks in Sec. XI. We conclude in Sec. XII.

II. Related work

While there is a very large literature on many aspects of bug reporting (see [14], [15] for literature reviews), there is a lack of literature measuring the influence of the bug reporting environment on the bug reporting activity or on any other aspects of software development. To our knowledge, there is no previous work measuring the impact of a change in bug reporting environment. More generally, there is little literature that compare bug trackers.

A. Comparing and proposing features of bug trackers

A simple, but quite limited, way to compare bug trackers is to look at the features that each of them proposes.

Karre et al. [16] compared 31 bug trackers feature-wise, and gathered these tools in four clusters. Bugzilla belongs to the cluster of bug trackers with many features (together with RedMine and Mantis), while GitHub belongs to a cluster of bug trackers attached to source code management systems (together with Savannah and BitBucket).

In the case of the Coq project, the loss of some specific features from Bugzilla was one of the arguments against the switch. Nonetheless, GitHub has some features that Bugzilla hasn't, and it is clearly not the number of features that determines which software is better or more adapted to the specific use case.

Additionally, Karre et al. recommended some missing features, including better support for migrating bug tracking data from a system to another. Our work has contributed to improve the situation in that sense. Nonetheless, it would be preferable if there existed a standard structured format for storing bug tracking information and all major bug tracking systems supported exporting to and importing from this format.

Abaee and Guru [17] listed the features of four commercial bug trackers, and proposed their own bug tracker with some new features, but without any evaluation.

There is much more literature proposing new features to add to bug trackers, but they are rarely evaluating the impact of adding such features on the bug tracking activity, even when a prototype was presented.

Baysal et al. [18] identified the need for personalized issue tracking systems after interviewing twenty Mozilla developers. They proposed a Bugzilla extension to address this need, and gave a, mostly qualitative, assessment by interviewing developers using their tool in [19].

Bortis [20] developed a bug triaging tool and evaluated how users interacted with it. However, there was no evaluation of its impact in the context of an actual software project.

Just et al. [21] surveyed developers from three large open source projects (Apache, Eclipse, and Mozilla; all three projects use Bugzilla) to identify missing features and provided recommendation to design better bug tracking systems.

Our article goes beyond a descriptive or normative perspective and quantitatively measures the causal impact of the change of the bug tracking platform (a crucial part of the bug tracking environment) on various aspects of bug reporting activity.

B. Analysis of projects' bug tracking data

Another way of comparing bug tracking systems could have been to conduct large-scale studies of many software projects using various bug trackers and provide some clues of the differences induced by the use of the different bug tracking systems. However, there are no such comparative studies in the literature.

Sowe et al. [22] noted that most preceding literature had only been comparing few projects at a time, usually using a single bug tracking system. In their study, they addressed this in part by studying hundreds of projects, but they did not mention which bug trackers the projects they compared used.

Bissyand et al. [23] published the same year a larger scale investigation using ten thousand projects' bug tracking data, analyzing such things as the correlation between a project's success and its bug tracking activity. All of the projects studied were using the same bug tracker (GitHub's).

Francalanci and Merlo [24] analyzed the bug fixing process by studying closed bugs from nine open source projects (four using JIRA, four using SourceForge's bug tracker, and one using another bug tracker). They did not, however, try to correlate the bug tracking activity with the bug tracking system used.

C. Switching developer tools

While, to the best of our knowledge, there is no work evaluating the impact of switching bug trackers, some studies have been conducted to evaluate the impact of switching other developer tools. Since bug trackers are important developer tools, our article also relates and contributes to this literature.

De Alwis and Sillito [25] surveyed developers to understand the perceived impact of moving from centralized to decentralized version control systems. As with bug trackers, the move is decided because of anticipated benefits: among them is the openness to external contributors. But the switch incurs challenges in the migration process (in particular, developers are very concerned to preserve the full history of the project) and some features are lost in the new tool (in particular, the

monotonically increasing revision numbers). Mulu et al. [26] performed a similar study based on interviews and a survey. They identified expectations and barriers and provided recommendations to developers wanting to conduct such a migration.

Squire [27] measured the impact of switching developer support channels from mailing lists and self-hosted forums to Stack Overflow, by first identifying the expectations of 20 projects which had officially decided on such a move, then by studying the impact on developer participation and response time for a selection of four such projects. On the other hand, Vasilescu et al. [28] compared the behavior of the same users on the R mailing list and the Stack Overflow platform. Contrarily to a bug tracker which is expected to be centralized for a given project, support channels can be diverse. Project leaders can decide to abandon their mailing list to focus exclusively on Stack Overflow, but they do not decide to "open" a support channel on Stack Overflow: it can exist nevertheless and the switch can be organic.

III. Context

A. Coq and GitHub

Coq is a (free and open source) proof assistant, i.e., a software to write and automatically verify mathematical proofs (and programs). It originated as a research project in 1984 and the development team is still composed, for the most part, of researchers. It gained in popularity along the years, and is now at the center of many verification projects (some very large academic projects, and some industrial projects). Its initial developers were awarded the ACM SIGPLAN Programming Languages Software 2013 award and the ACM Software System 2013 award.

The development is currently centered around the GitHub platform. The Coq repository counts more than 25,000 commits. Some of these commits date back from 1999, when the code received major architectural changes. The GitHub repository was initially created as a mirror of the official repository (which itself was migrated from SVN to git in 2013) and really started attracting the interest of the development team as a way to open up the development to external contributions, especially starting with the first Coq coding sprint in 2015 (later renamed into the Coq Implementors Workshop and held every year in France). From then on, core developers started using pull requests for their own changes more and more often, and it became the norm around the beginning of 2017 when systematic continuous integration tests were introduced on every pull request. In the last two years, more than 2,200 pull requests have been opened.

B. The Coq bug tracker

From 2007 to 2017, the Coq bug tracker platform was a self-hosted Bugzilla instance (before this, from 2001 to 2007 it was using JitterBug, a now discontinued bug tracking system, and before 2001 a simple mailing list). In 2017, developers had become quite accustomed to the look and feel of discussion threads on GitHub pull requests, which offered an arguably nicer experience than the Bugzilla bug tracker. Younger developers were frustrated by what they perceived as an older and heavier platform. The rest of the developers were convinced to migrate towards a non-self-hosted solution in August 2017, when the whole Coq website was shut down for more than a week because an IT team was frightened by a scam bug report.[1]

The expected advantages of the GitHub issue tracker over Bugzilla were:

- Better formatting, ability to edit comments, and more generally a more pleasant tool to use. According to some Coq developers, Bugzilla was unpleasant enough to discourage developers to use it to its full potential.
- Version control integration: Browsing between code, pull requests and bug reports is easier when everything is in the same place. Furthermore, it means a common notification, mention, and assignment system.
- Better indexing by search engines.
- Shared management of permissions, milestones, categories (labels), etc.
- Easier to get started for newcomers (especially as many may already know GitHub and already have a GitHub account).

The possible drawbacks were mostly that GitHub provides a less advanced system with very few customization options. In particular:

- The permission system is much less fine-grained. Giving triaging rights to users requires giving them "write access", which includes commit rights on non-protected branches. That's why the Coq GitHub repository now only has protected branches (and accidentally-pushed topic branches are promptly deleted).
- Creating a new bug report does not require filling a form, but typing a message. Fortunately, issue templates help give some hints towards the expected structure of the report, and being able to strip this template can actually give flexibility to developers who might open new issues for concerns that do not qualify as bugs.
- There is no import mechanism that allows to keep the number and author of preexisting reports. Fortunately, as we will see in Sec. V, it was still possible to import bug reports while keeping most of the meta-information and renumbering the fewest possible bug reports. This was an important requirement for the switch.

IV. Hypotheses

Given the advantages of the GitHub issue tracker that were listed above, one could expect the switch to:

- H1 Increase the bug tracking activity, in terms of the number of reported bugs;
- H2 Extend the pool of bug reporters and commentators;
- H3 Increase the bug tracking activity, in terms of the amount of interaction between users and developers.

All of this could lead in turn to a greater openness and transparency of the development and an increased engagement

[1] https://sympa.inria.fr/sympa/arc/coq-club/2017-08/msg00040.html

of users toward it, changing (and probably improving) the dynamics of a crucial part of software development.

V. Migration

The only complicated part of the bug tracker switch was the migration of preexisting bug reports. We reused a tool by Andriy Berestovskyy[2] which is designed to import Bugzilla reports (extracted as an XML dump) to GitHub using its REST API. The bugs are imported in an order designed to preserve numbers whenever possible. Bugs whose number is unavailable (e.g. because the number is already taken by a GitHub pull request) are postponed and renumbered. We implemented several improvements to make the tool better fit the needs of the Coq project:

- *Allowing non-consecutive bug numbers:* The imported set of bug reports had some holes in the numbering due to deleted bugs. We use postponed bugs to fill the holes. This improvement has now been integrated upstream.
- *Saving a table of correspondence for renumbered bugs:* This was used later to redirect the old bug URLs to the new ones.
- *Using the GitHub issue import beta API and overcoming the GitHub rate limits:* Creating a new issue or a new comment through the normal GitHub REST API will trigger notifications (for people who are watching the repository or are mentioned in the issue thread). Therefore, GitHub chooses to impose a strict rate limit on these actions, which prevented using this tool for importing more than a few hundred bugs. Fortunately, GitHub provides a (beta) issue import API which, in addition to not triggering notifications, also allows importing one bug, its comments, and meta information such as closed status and assignee in a single request (thus reducing the duration to import 4900 bugs to just a few hours). Furthermore, using this API allows to keep the dates of imported bug reports and comments, which is very useful to this study. We didn't manage to import correct closing dates for every bug report, so we will not study the impact of the bug tracker switch on the time it takes to close a bug.

  This improvement has now been integrated upstream as an optional setting.

This tool required a correspondence table between Bugzilla and GitHub accounts. Ours was created by manually matching 217 (out of 686) Bugzilla accounts to 175 GitHub accounts,[3] the difference being due to the high prevalence of duplicate Bugzilla accounts. We gave priority to finding GitHub accounts for people who still had opened bug reports.

Once the switch was approved by the development team on October 4th, 2017, the migration had to happen as soon as possible because every new pull request before the migration added to the number of bug reports that would need to be renumbered. It was conducted on October 18th, the day after the 8.7.0 release (and not before to avoid disturbing the release process). Only 502 out of 4900 bugs (whose numbers were below 1154) had to be renumbered. This number is quite low because these bugs are dating back from the JitterBug era and during the previous switch only the bugs that were still open had been migrated. Due to the rate of pull request creation, if the switch had been delayed by a year, the number of renumbered bugs would have been closer to 3000.

VI. Data

A. Extraction

All the data for this study was extracted on January 17th, 2019 using the GitHub GraphQL API [29]. Using this API allows to do large requests (100 nodes in a single request) with just the information we need (thus both reducing the bandwidth usage and speeding up the extraction process). We provide the extracted data as CSV files and a Jupyter notebook[4] with the code to request this data from GitHub, to load the CSV files, to run the pre-processing steps, and the analyses.

Migrated bugs and comments have their author information stored in the text, we extract it from there. Dates of bug and comment creation have been preserved which allows us to obtain them transparently for bugs before and after the migration. Bug reports and comments from the JitterBug era are missing information and were not fully migrated, therefore we do not consider any data from before 2008.

B. Pre-processing

a) Excluding specific reporters: We have one specific reporter, Jason Gross, who is alone responsible for almost a quarter of all bug reports. To avoid having the behavior of a single individual strongly impact the overall statistics, we exclude his comments, bug reports and the comments they received from our analysis. Similarly, we also exclude the developer who was the main advocate of the bug tracker switch as it could have influenced his behavior before and after the switch.

b) Merging duplicate Bugzilla accounts: We found that a significant number of users had created several accounts on the Bugzilla system. We merged them to avoid overestimating the rate of new reporters which is analyzed in the companion Jupyter notebook.

c) Removal of migration artifact comments: The migration tool created an issue whose body contains only meta-information, and a first comment with the description, for every bug report. These comments are thus migration artifacts and we remove them from our comment analyses. They can be easily identified because they are the comments that were posted at the exact same date and time as the corresponding bug report.

[2]Tool available at https://github.com/berestovskyy/bugzilla2github.

[3]The correspondence table can be seen at https://frama.link/vRV9etCd.

[4]https://github.com/Zimmi48/impact-of-switching-bug-trackers.

C. Variable definition

We measure different indicators of bug tracker activity: the numbers of bug reports per day; the number of distinct bug reporters in a week; the number of new bug reporters (who had never reported a bug before) per day; the number of comments per day; the number of distinct commentators in a week; and the number of new commentators per day. The number of distinct bug reporters and commentators is intended to allow us to distinguish between having a few prolific contributors and a large base of casual contributors. We use an interval of a week instead of a day for measuring the number of distinct bug reporters and commentators because, at the scale of a day, there is less opportunity for repeated contributions by the same contributor, but longer scales would compromise the estimation of effects by removing too much data.

For the visualization of Fig. 5 to 8, we plot a point for the average of each two week period. This is just to allow an easier and clearer visual analysis of the discontinuity. The regression lines, however, correspond to the definitions detailed in the above paragraph (i.e., analysis with daily periods or weekly periods depending on the variable).

We analyze heterogeneous effects by distinguishing between developers and other contributors (others being assimilated to "users"). We define "developers" as the persons who have contributed more than 100 commits since 2008. We identified 18 developers. These developers are responsible for 91.5% of all commits since 2008 (this is consistent with standard results on the proportion of commits by the "core team" in open source software [30]).

For the dates of the new releases in Section VII, we define release dates as the dates of the release announcement on the mailing list or of the corresponding news item on the official website (https://coq.inria.fr/news), whichever comes first. We selected beta and major releases and excluded release candidates and patch-level releases. The first release to appear in the news section of the current official website is the 8.2 beta release in June, 2008.

VII. Descriptive statistics

The companion Jupyter notebook contains an extensive array of descriptive tables and figures, such as the distribution of bug reporting in the year and in the day, graphs of bug reporting for individual developers, etc. We only present a subset of these results.

A. Cumulative number of bug reports and bug comments

Fig. 1 and 2 give a first (cumulative) view of two of our outcome variables: the number of bug reports, and the number of comments. From these figures, we can clearly notice the dominance of developers in the number of bug comments and the dominance of non-developers in the number of bug reports. We can also already notice some slope changes at the bug tracker switch date. The magnitude and significance of this changes will be assessed in Sec. IX.

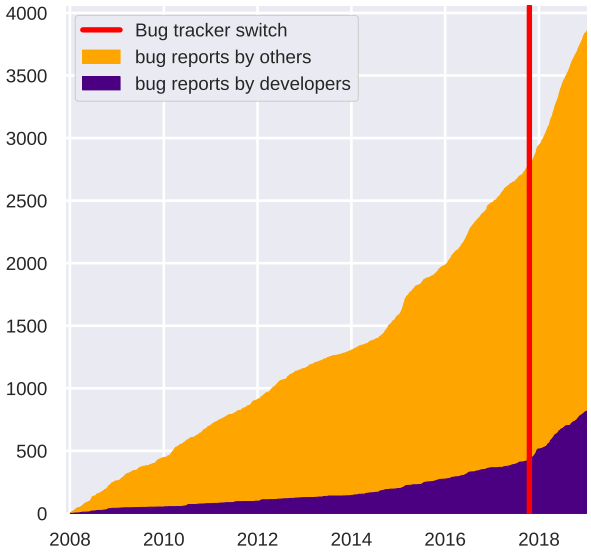

Fig. 1. Cumulative number of bug reports (since 2008).

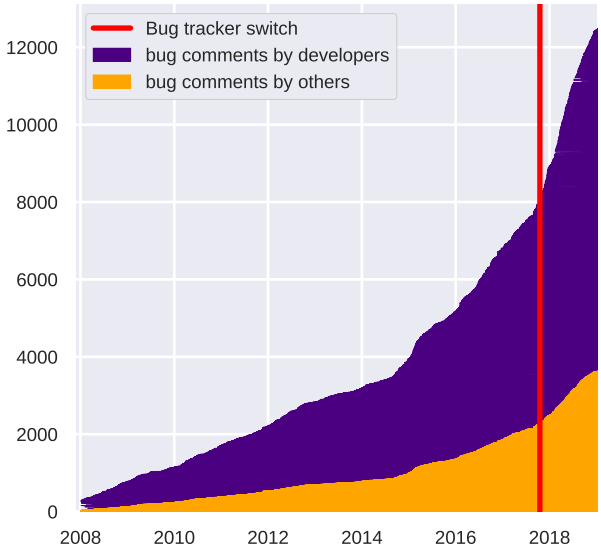

Fig. 2. Cumulative number of comments (since 2008).

B. Influence of new releases

a) Influence on the number of bug reports: We expect release dates to be correlated to peaks of activity on the bug tracker, in particular peaks of new bug reports. Indeed, users will generally test new releases, for instance by trying to port preexisting projects to the new version, which might lead them to find new bugs (either regressions or bugs related to new features). In Fig. 3, we can see that while release dates are generally correlated to peaks of new bug reports by non-developers, this is not so much the case of reports by developers. We can also notice that beta releases are often correlated to higher peaks than final releases which is not surprising given that the users who test beta versions are more likely to be ready to report bugs, and beta versions are likely

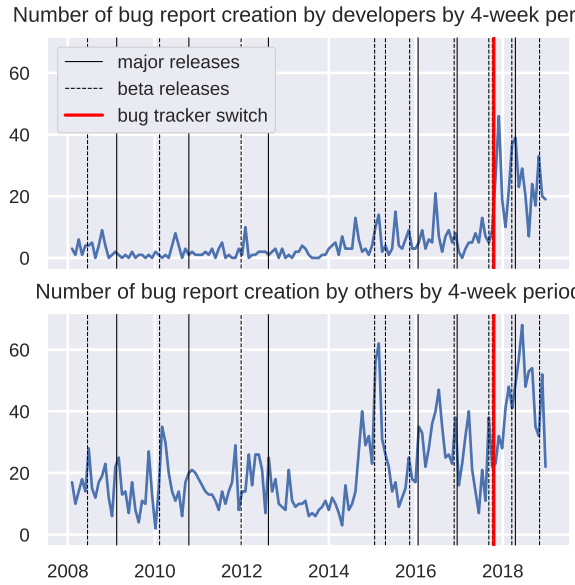

Fig. 3. Number of bug reports by 4-week period with release dates (since 2008).

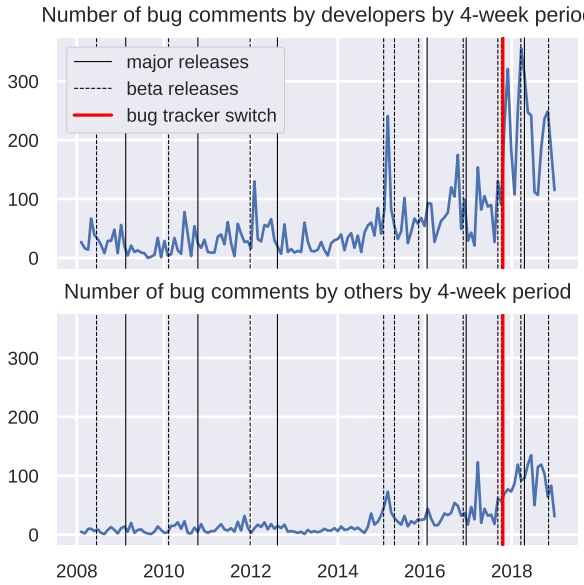

Fig. 4. Number of bug comments by 4-week period with release dates (since 2008).

to be less polished.

The highest peak is in 2015, after the first beta release of version 8.5. This was the first release in more than two years and it contained five big new features ("the result of five specific long-term projects"[5]). Stabilizing these features and their (unplanned) interactions required more than a year of testing and three beta releases. The 8.5 release was both hard on developers, who found the release cycle too long, and on users, who were impacted by many compatibility issues. Consequently, a different approach was taken for the following releases: versions 8.6, 8.7 and 8.8 were "developed on a time-based development cycle" and contain "the result of refinements, stabilization of features and cleanups of the internals of the system along with a few new features". Attention was given to regression testing: changes were systematically tested by building a selection of the largest Coq projects. Therefore, it is not surprising that these frequent releases are correlated to smaller peaks of new bug reports.

b) Influence on the number of comments: In Fig. 4, we can see that release dates are correlated to peaks of comments by developers (and not so much by other people). This is explained by the fact that bug reports are generally answered by developers, thus any peak in bug reporting is bound to induce a peak in bug commenting by developers.

VIII. Methodology

We exploit the fact that the switch from Bugzilla to GitHub can be seen as a clear cutoff in time to use a temporal sharp Regression in Discontinuity design (RDD) to estimate the effects of the switch on our outcome variables; to determine if these estimated effects are statistically significant; and to interpret these effects in a causal way. The idea behind RDD is that factors that influence bug reporting activity are fairly similar just before and just after the cut-off point, which makes the changes observed between just after and just before the cut-off a consequence of the change that occurred at the cut-off (i.e., the switch of bug reporting platform).

The assumption behind this analysis is that bug reporters did not choose the exact moment in which the switch would take place. In the absence of any other discontinuity, the changes in the behavior of bug reporters just before and just after the migration is likely to depend mainly on the switch: the other factors influencing bug reports evolve slowly, and should be pretty similar just before and just after the switch. It is important to note that estimated effects using this method will inform of the effect near the threshold (i.e., it will only give the Local Average Treatment Effect –LATE– of the switch). Indeed, in the same way that behavior just before and just after the threshold is not likely to be different in the absence of the switch, the behavior several periods before and several periods after is likely to be different even in the absence of the switch. See [11] and [10] for an accessible and intuitive introduction to Regression on Discontinuity analysis; and [13] and [12] for further details and a practitioner's guide to its empirical application.

We model the evolution of the bug reporting behaviour around the switch by two different functions, one before and one after the switch, fitted using ordinary least squares. Their purpose is to accurately estimate the value at the switch in the absence and presence of treatment. Since only their value around the cutoff point is relevant, we can choose to estimate them on a restricted bandwidth (number of periods) around

[5]Quote from:
https://coq.inria.fr/distrib/V8.9.0/refman/credits.html#credits-version-8-5
The next quotes are coming from the following sections of the same chapter of the Coq reference manual.

the cutoff, or on a larger dataset. We also have the choice of different functional forms for these models.

In the context of a Regression in Discontinuity analysis, the choice of the bandwidth of the analysis and the functional form of the regression is always difficult. On the one hand, including data points that are far from the cut-off (in our case the switch) will increase precision as it will reduce standard errors (because we will have more data points). On the other hand, however, including those points can bias estimation near the cut-off if the functional form of the evolution in the absence of switch is not accurately specified. Our main specification takes a conservative approach with a relatively small bandwidth of 180 days before and after the switch to minimize possible bias, and a comparatively simpler linear model. However, as a robustness check, we also estimate a quadratic model on a larger time frame (455 days on each side). This conservative approach reduces the statistical power of our analysis, increasing the probability of observing false negatives (being unable to detect an effect where such an effect exists).

We use time as rating variable[6] (with the cutoff centered at zero) and we estimate the following regression:

$$\begin{aligned} &\textit{Number of bug reports}_t = \\ &\quad \gamma_0 + \gamma_1 \times \textit{Relative date}_t + \gamma_2 \times \textit{After switch}_t \\ &\quad + \gamma_3 \times \textit{Relative date}_t \times \textit{After switch}_t + \epsilon_t \end{aligned}$$

where $\textit{Number of bug reports}_t$ is the total number of bugs reported during the day t; $\textit{Relative date}_t$ is the number of days from the date of the switch (zero is the first period after the switch); $\textit{After switch}_t$ is a binary variable equal to one if t is a period after the switch and zero otherwise; $\textit{Relative date}_t \times \textit{After switch}_t$ is called the interaction term and ϵ_t is the residual error. This is equivalent to estimating the following two regressions, respectively before and after the switch:

$$\begin{aligned} &\textit{Number of bug reports}_{t<0} = \\ &\quad \gamma_0 + \gamma_1 \times \textit{Relative date}_t + \epsilon_p \end{aligned}$$

$$\begin{aligned} &\textit{Number of bug reports}_{t\geq 0} = \\ &\quad (\gamma_0 + \gamma_2) + (\gamma_1 + \gamma_3) \times \textit{Relative date}_t + \epsilon_t \end{aligned}$$

We estimate this regression by Ordinary Least Squares [31] and compute heteroscedasticity robust standard errors.[7] Coefficient γ_0 is the estimated value just before the cutoff and γ_1 the slope before the cutoff. Coefficients γ_2 and γ_3 are the estimates of interest and will tell us the jump of the number of bugs reports just after the switch and the change in slopes due to the switch respectively.

We estimate this regression for two different sub-samples: the developers and the non-developers. We replicate this analysis for all our outcome variables by changing the variable on the left hand side of the equation.

[6]The rating variable is also sometimes called the running variable or the forcing variable in the literature.

[7]The errors ϵ_t are said to be heteroscedastic if their variances differ. The standard calculations for estimation errors assume equal variances [31].

TABLE I

ESTIMATED IMPACT OF THE SWITCH ON THE NUMBER OF BUGS. COEFFICIENTS ARE HIGHLIGHTED IF THEY ARE STATISTICALLY SIGNIFICANT WITH $p < 0.1$ (†), $p < 0.05$ (*), $p < 0.01$ (**) OR $p < 0.001$ (***). STANDARD ERROR IS IN PARENTHESES.

	Total	Developers	Others
After switch$_p$	0.648†	0.7**	-0.0525
	(0.35)	(0.234)	(0.241)
Relative date$_p$	0.00143	-0.000393	0.00182
×*After switch*$_p$	(0.00328)	(0.00214)	(0.00234)
Relative date$_p$	0.00276	0.000133	0.00263†
	(0.00174)	(0.000755)	(0.00152)
Constant	1.21***	0.284***	0.927***
	(0.207)	(0.0789)	(0.179)
Observation number	360	360	360

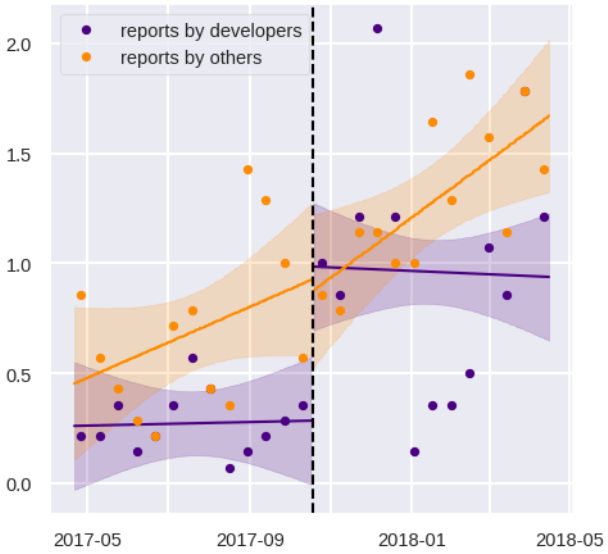

Fig. 5. Average number of bug reports per day before and after the migration (with fitting lines and confidence intervals from the regression results).

IX. RESULTS

A. Impact on the number of bug reports

Table I presents the estimated impact of the switch on the number of reported bugs. Each column shows the estimates for a different sub-sample (i.e., all bug reporters, developers and non-developers). Asterisks represent different classical levels of statistic significance. We interpret results as significant if the p-value is below 0.1. Estimates that are not statistically significant cannot be interpreted as an absence of effect but indicate that if such effect occurred, we are unable to discern it with our conservative approach. Fig. 5 shows the number of reported bugs before and after the switch and the fitting lines and confidence intervals corresponding to the regression results.

For developers, we see a statistically significant positive jump in bug reports just after the switch. In particular changing the bug reporting platform is estimated to have increased the daily number of reported bugs by 0.7 (representing an increase of around 250%). That is, on average, we observe around one bug reported by developers every day after the switch, while

TABLE II
ESTIMATED IMPACT OF THE SWITCH ON THE NUMBER OF DISTINCT REPORTERS EACH WEEK. COEFFICIENTS ARE HIGHLIGHTED IF THEY ARE STATISTICALLY SIGNIFICANT WITH $p < 0.1$ (†), $p < 0.05$ (*), $p < 0.01$ (**) OR $p < 0.001$ (***). STANDARD ERROR IS IN PARENTHESES.

	Total	Developers	Others
After switch$_p$	2.17	1.92*	0.245
	(1.68)	(0.977)	(1.27)
Relative date$_p$	0.146	0.0162	0.13
×*After switch*$_p$	(0.108)	(0.0648)	(0.0862)
Relative date$_p$	0.0608	-0.00462	0.0654
	(0.0903)	(0.0378)	(0.0675)
Constant	6.03***	1.5**	4.53***
	(1.47)	(0.579)	(1.11)
Observation number	50	50	50

TABLE III
ESTIMATED IMPACT OF THE SWITCH ON THE NUMBER OF COMMENTS. COEFFICIENTS ARE HIGHLIGHTED IF THEY ARE STATISTICALLY SIGNIFICANT WITH $p < 0.1$ (†), $p < 0.05$ (*), $p < 0.01$ (**) OR $p < 0.001$ (***). STANDARD ERROR IS IN PARENTHESES.

	Total	Developers	Others
After switch$_p$	5.5*	4.76**	0.739
	(2.24)	(1.79)	(0.717)
Relative date$_p$	0.0155	0.0144	0.00112
×*After switch*$_p$	(0.0205)	(0.017)	(0.0063)
Relative date$_p$	0.00147	-0.00288	0.00435
	(0.00882)	(0.00673)	(0.00383)
Constant	4.77***	2.95***	1.82***
	(1.03)	(0.704)	(0.463)
Observation number	360	360	360

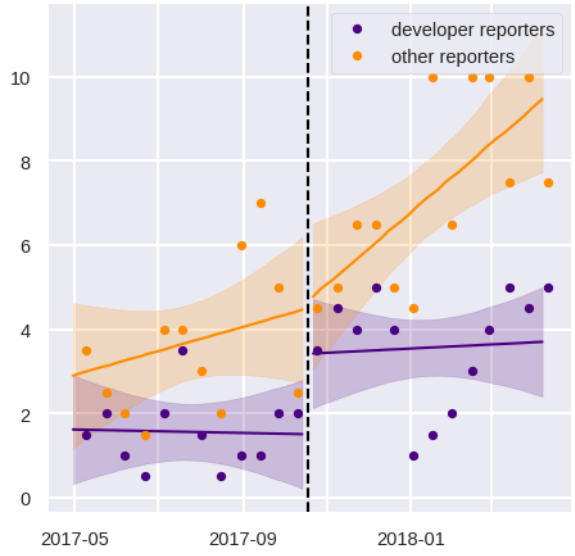

Fig. 6. Average number of distinct reporters each week before and after the migration (with fitting lines and confidence intervals from the regression results).

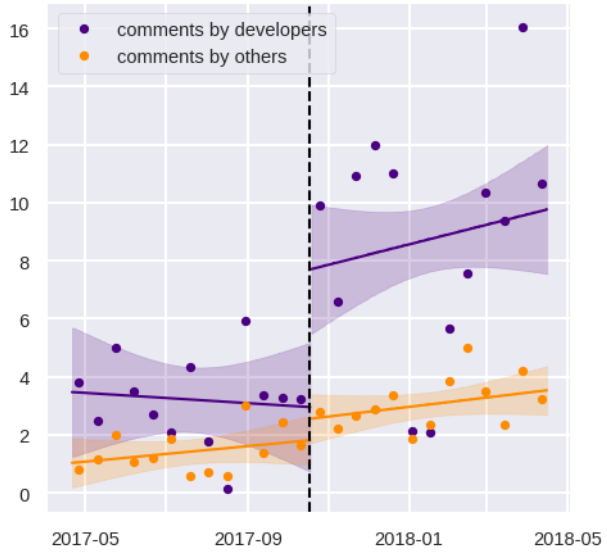

Fig. 7. Average number of bug comments per day before and after the migration (with fitting lines and confidence intervals from the regression results).

before the switch developers reported a bug every three days. There is no statistically significant effect on the number of bug reports by other contributors: we cannot reject that such a variation occurred, but we are unable to discern it with our conservative approach.

B. Impact on the number of distinct bug reporters

Table II and Fig. 6 show the estimated impact of the switch on the number of distinct bug reporters each week. These results demonstrate that the switch to GitHub positively affected not just the number of bug reports but also the number of distinct developer bug reporters in a given week. The number of distinct developers that reported bugs in a given week increased by around 130% (from a 1.5 reporting developers each week before the switch to 3.42 after)

C. Impact on the number of comments

Table III and Fig. 7 show the estimated impact of the bug tracker switch on the number of comments. We see a statistically significant jump in the number of comments, which more than doubles (from 4.8 comments per day before the switch to 10.3 after the switch). This appears to be, for the most part, due to comments by developers who increased their average number of comments per day from around 3 before the switch to around 8 after.

D. Impact on the number of distinct commentators

Table IV and Fig. 8 show the estimated impact of the bug tracker switch on the number of distinct commentators each week. We can see a statistically significant jump in the total number of commentators, in the number of developer commentators, and in the number of non-developer commentators.

In general terms, the number of distinct commentators in a given week changed from an average of 7 to an average of 13 (almost doubling the commentators). Analyzing the developers and the non-developers, we can see that this increase is due in larger part to non-developers (whose number of commentators more than doubles) while, among developers, this number increases by 66%.

TABLE IV
ESTIMATED IMPACT OF THE SWITCH ON THE NUMBER OF DISTINCT COMMENTATORS EACH WEEK. COEFFICIENTS ARE HIGHLIGHTED IF THEY ARE STATISTICALLY SIGNIFICANT WITH $p < 0.1$ (†), $p < 0.05$ (*), $p < 0.01$ (**) OR $p < 0.001$ (***). STANDARD ERROR IS IN PARENTHESES.

	Total	Developers	Others
After switch$_p$	6.19*** (1.81)	2.53* (1.05)	3.66** (1.17)
Relative date$_p$ ×*After switch*$_p$	0.205† (0.118)	0.0769 (0.0641)	0.128 (0.0924)
Relative date$_p$	-0.0692 (0.0833)	-0.0469 (0.0589)	-0.0223 (0.0463)
Constant	7.06*** (1.54)	3.83*** (0.984)	3.23*** (0.861)
Observation number	50	50	50

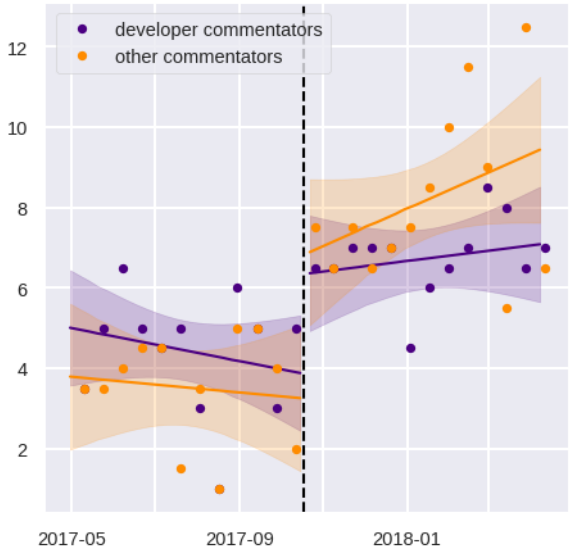

Fig. 8. Average number of distinct commentators each week before and after the switch (with fitting lines and confidence intervals from the regression results).

X. DISCUSSION

A. H1: impact on the number of reported bugs

We hypothesized that the switch to a platform that is perceived as more modern and easier to use would increase the number of reported bugs. Our data analysis only shows such an effect on the number of bug reported by developers, and not on the bugs reported by non-developers, which only partially validates our first hypothesis.

The developers were using the bug tracker very little before the switch. This means that when they found bugs in the course of the development, they stored them in personal task lists, fixed them immediately, or tried to remember them in their heads until they could fix them (likely with some bugs being forgotten in the process). Meanwhile, they were discussing a lot through the pull request system on GitHub. The move of the bug tracker to GitHub made it closer to the main development platform (the repository where pull requests are submitted and reviewed), both in terms of location and in terms of user interface. These reasons are probably what pushed developers to use it more.

The increased use of the bug tracker in place of personal task lists makes the whole development process more transparent and is likely to be beneficial to new contributors.

If it is the case that the switch had no effect on the number of bugs reported by non-developers, this could mean that when people decide to report bugs, they do so before accessing the bug tracker, and nor the requirement of creating an account, nor the heavier and less modern interface, are sufficient to discourage them.

B. H2: impact on the pool of reporters and commentators

Having a common platform for the pull requests and the bug reports, and using a standard solution where a lot of users would already have an account, was hypothesized to have an impact on the pool of participants in the bug tracker. For non-developers, our data analysis only shows this effect in a statistically significant way on the number of commentators, which only partially validates our second hypothesis.

Analyzing deeper, we find that there exists not just a jump in the non-developer commentators in a given week but also a jump in the number of new non-developer commentators (i.e., people that had never before posted a comment). We show the results for the new commentators in the companion Jupyter notebook.

While we suggested that reporters would be motivated enough to not be affected by the bug tracking platform, this is less likely to be the case for people willing to comment. Furthermore, people who were already following the pull request activity were now subscribed to the bug tracking activity as well and this could have been enough to encourage them to post comments. The previous system made it harder and less flexible to subscribe to the bug tracker activity.

This analysis is consistent with previous findings: many people start by *lurking* (i.e., mostly reading discussions) before starting to contribute to an open source project (for instance by participating to discussions) [32]. In the context of GitHub, a significant portion of the new contributors start by *watching* the repository (i.e., subscribing to its activity). These contributors contribute for longer periods and achieve a wider variety of contribution types, probably due to the knowledge and confidence they gained during the watching process [33].

The significant impact on distinct reporters and commentators among developers is more surprising but shows that some developers were disengaged from any kind of activity on the bug tracker before the switch. One developer explained this as a difficulty to keep track of activity on two platforms at once (GitHub for the pull requests and Bugzilla for the bug reports). Comparing data from pull request commentators and bug report commentators confirmed this: before the switch there were more developers commenting pull requests than bug reports in any given week, whereas after the switch this difference disappeared completely (graph available in the accompanying notebook).

C. H3: impact on the interaction between users and developers

The surge in the number of comments posted by developers shows that developers communicate more on the bug reports they receive, instead of just fixing and closing them. This result suggests once again that the switch was beneficial in terms of transparency of the development, an evolution that is likely to give greater satisfaction to the bug reporters and probably encourage engagement in the process of bug reporting and development more generally. Taken together with the increased number of non-developer commentators, this validates our third hypothesis.

Overall, these results suggest that the change of platform reinforced discussion among developers and among developers and non-developers, which may improve efficiency in the developing process, incentivize non-developers to have a more active role (maybe even assuming developing tasks) and in the long run increase the quality of the software (as bugs are probably better understood and fixed in better accordance to users' needs).

XI. Threats to validity

A. Internal validity and robustness checks

The biggest concern that we have in our analysis is the fact that the date of the switch is close to the release of a new version of Coq. The release of version 8.7.0 took place on October 17th, and the migration on October 18th. Then, the hypothesis that there is no other discontinuity around the threshold of interest does not completely hold in our case. To confirm that this is not what is driving our results, we conduct two robustness checks with our data. The first consists in removing the two weeks following the 8.7.0 release (and the switch) from our Regressions in Discontinuity. With these periods removed, the regressions still give very close results. The second consists in including a binary control variable that is equal to one if the period is in the two weeks after a release and zero otherwise. This control variable will capture the high peaks due to the release without the need of eliminating periods close to the switch. Again, our results remain very similar when we include this control variable.

Our main specification for Regression on Discontinuity uses a small bandwidth and a linear model. In this section, we also present the results using all the available data (455 days on each side of the cutoff, corresponding to all the data that we collected after the switch and the same number of days before). To reduce the associated bias, we include a quadratic term to allow the regression to better fit the data and "adjust" to far away data points. Following Gelman et al. [34], we include only a quadratic polynomial and not a polynomial of a higher order. Our results still hold with this specification.

B. External validity

This is only a case study on a very specific project (by the target audience, the location and composition of its core team, etc.). We would really benefit from conducting similar analyses on other projects that went through a similar bug tracker switch.

We located another project that went through a similar switch from Bugzilla to GitHub and used the Bugzilla import tool with our modifications. It would have been a perfect case for replicating this analysis. A preliminary visualization of the data shows that, compared to the few years before the switch, there have been more bug reports and comments posted after the switch. Unfortunately, we learned that some data was lost during the crash of their previous bug tracker, which makes it really difficult to draw any reliable conclusion.

We nonetheless believe that our conclusions can be generalized to some extent and encourage researchers to replicate this study, for instance using our migration tool and/or our analysis pipeline and methodology.

There is no reason to believe that these results would generalize to proprietary software projects. Some projects that are also less likely to benefit as much from a switch to GitHub are the larger projects (like Mozilla, Linux, Gnome, KDE, Apache) which have been using Bugzilla for a long time and have defined rigorous development processes, in particular regarding the use of the bug tracker. Coq is an example of a project that is developed in a very Bazaar-like fashion, with no clear rules on how to use the bug tracker, and this lack of rules was hindering the use of a complex bug tracker like Bugzilla. We believe that open source projects that share this lack of rules and this medium size are the more likely to benefit from a switch from Bugzilla (or similar bug trackers like Mantis), to GitHub (or the very similar GitLab platform), especially if they are already heavily using the latter for pull requests.

XII. Conclusion and perspectives

We have analyzed the impact of changing the bug reporting environment using a Regression in Discontinuity analysis with data from the switch of the Coq bug tracker from Bugzilla to GitHub. We have shown that the switch incurred an increase in bug tracking activity in two ways. First, the developers are using the bug tracker more, to report their own bugs, and to discuss bug reports. Second, the non-developers are more numerous to take an active role by commenting bug reports as well. We find particularly interesting the change in the number of comments by developers and in the number of distinct non-developer commentators because it shows a shift in the dynamics of the bug reporting process that we believe can improve software development by engaging more people in the process, and by increasing openness and transparency.

Given the lack of literature on the effects of bug tracking environments, interested researchers could expand our analysis in many different ways. Analyzing other outcomes (such as pull requests or the rapidity of bug resolution) could give interesting additional insights and help create a more complete picture of the mechanisms at play. Replicating our analysis for different contexts and software would increase the external validity of the results.

Finally, we introduced and showed an example of the use of a Regression on Discontinuity design, that is novel in the

context of empirical software engineering. We think that there are many cases where this methodology would be useful to help demonstrate causality from mined software repository data.

Supporting data. We provide our extraction and analysis code as a Jupyter notebook and the data we used as CSV files. We also provide the XML export of Bugzilla that was used for the migration and the migration script. Everything is available from https://github.com/Zimmi48/impact-of-switching-bug-trackers.

Acknowledgements. We would like to thank Ambre Williams for valuable advice and help on the design and implementation of the data analyses and Cécile Rottner for general advice, support, and proof-reading. We would like to thank Andriy Berestovskyy for providing the Bugzilla import tool, and Martin Michlmayr for doing the first steps in porting our modifications upstream.

REFERENCES

[1] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[2] D. W. van Liere, "How shallow is a bug? why open source communities shorten the repair time of software defects," in *Proceedings of the 30th International Conference on Information Systems*, 2009, p. 195.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM, 2005, pp. 35–39.

[4] J. L. Davidson, N. Mohan, and C. Jensen, "Coping with duplicate bug reports in free/open source software projects," in *Proceedings of the 2011 Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2011, pp. 101–108.

[5] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful really?" in *Proceedings of the 24th International Conference on Software Maintenance*. IEEE, 2008, pp. 337–345.

[6] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in *Proceedings of the 2010 Conference on Computer Supported Cooperative Work*. ACM, 2010, pp. 301–310.

[7] C. Jensen and W. Scacchi, "Role migration and advancement processes in OSSD projects: A comparative case study," in *Proceedings of the 29th International Conference on Software Engineering*. IEEE/ACM, 2007, pp. 364–374.

[8] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the International Workshop on Principles of Software Evolution*. ACM, 2002, pp. 76–85.

[9] Y. Ye and K. Kishida, "Toward an understanding of the motivation open source software developers," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE/ACM, 2003, pp. 419–429.

[10] J. D. Angrist and J.-S. Pischke, *Mostly harmless econometrics: An empiricist's companion*. Princeton university press, 2008.

[11] ——, *Mastering'metrics: The path from cause to effect*. Princeton University Press, 2014.

[12] R. Jacob, P. Zhu, M.-A. Somers, and H. Bloom, "A practical guide to regression discontinuity." *MDRC*, 2012.

[13] D. S. Lee and T. Lemieux, "Regression discontinuity designs in economics," *Journal of economic literature*, vol. 48, no. 2, pp. 281–355, 2010.

[14] J. D. Strate and P. A. Laplante, "A literature review of research in software defect reporting," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 444–454, 2013.

[15] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 2016.

[16] S. A. Karre, A. Shukla, and Y. R. Reddy, "Does your bug tracking tool suit your needs? a study on open source bug tracking tools," *arXiv preprint arXiv:1706.06799*, 2017.

[17] G. Abaee and D. Guru, "Enhancement of bug tracking tools; the debugger," in *Proceedings of the 2nd International Conference on Software Technology and Engineering*, vol. 1. IEEE, 2010, pp. V1–165.

[18] O. Baysal, R. Holmes, and M. W. Godfrey, "Situational awareness: personalizing issue tracking systems," in *Proceedings of the 35th International Conference on Software Engineering*. IEEE/ACM, 2013, pp. 1185–1188.

[19] ——, "No issue left behind: Reducing information overload in issue tracking," in *Proceedings of the 22nd International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 666–677.

[20] G. T. Bortis, "Porchlight: A tag-based approach to bug triaging," Ph.D. dissertation, UC Irvine, 2016.

[21] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," in *Proceedings of the 2008 Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2008, pp. 82–85.

[22] S. K. Sowe, R. A. Ghosh, and K. Haaland, "A multi-repository approach to study the topology of open source bugs communities: Implications for software and code quality," *International Journal of Computer Theory and Engineering*, vol. 5, no. 1, p. 138, 2013.

[23] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *Proceedings of the 24th International Symposium on Software Reliability Engineering*. IEEE, 2013, pp. 188–197.

[24] C. Francalanci and F. Merlo, "Empirical analysis of the bug fixing process in open source projects," in *Proceedings of the 4th International Conference on Open Source Systems*. IFIP, 2008, pp. 187–196.

[25] B. De Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. IEEE, 2009, pp. 36–39.

[26] K. Muşlu, C. Bird, N. Nagappan, and J. Czerwonka, "Transition from centralized to decentralized version control systems: A case study on reasons, barriers, and outcomes," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 334–344.

[27] M. Squire, "" should we move to stack overflow?" measuring the utility of social media for developer support," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 2. IEEE/ACM, 2015, pp. 219–228.

[28] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social q&a sites are changing knowledge sharing in open source software communities," in *Proceedings of the 17th Conference on Computer Supported Cooperative Work & social computing*. ACM, 2014, pp. 342–354.

[29] *GitHub GraphQL API v4, GitHub Developer Guide*, accessed January 19, 2019. [Online]. Available: https://developer.github.com/v4/

[30] G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz, "Evolution of the core team of developers in libre software projects," in *Proceedings of the 6th International Working Conference on Mining Software Repositories*. IEEE, 2009.

[31] J. M. Wooldridge, *Introductory econometrics: A modern approach*. Nelson Education, 2015.

[32] G. Von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research policy*, vol. 32, no. 7, pp. 1217–1241, 2003.

[33] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian, and J. Ell, "Understanding watchers on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 336–339.

[34] A. Gelman and G. Imbens, "Why high-order polynomials should not be used in regression discontinuity designs," *Journal of Business & Economic Statistics*, pp. 1–10, 2018.