How Does the Shift to GitHub Impact Project Collaboration?

Luiz Felipe Dias[1] Igor Steinmacher[1] Gustavo Pinto[2] Daniel Alencar da Costa[3] Marco Gerosa[4]
[1]UTFPR, Brazil [2]IFPA, Brazil [3]UFRN, Brazil [4]USP, Brazil
luizdias@alunos.utfpr.edu.br, igorfs@utfpr.edu.br, gustavo.pinto@ifpa.edu.br, danielcosta@ppgsc.ufrn.br, gerosa@ime.usp.br

***Abstract*—Social coding environments such as GitHub and Bitbucket are changing the way software is built. They are not only lowering the barriers for placing changes, but also making open-source contributions more visible and traceable. Not surprisingly, several mature, active, non-trivial open-source software projects are switching their decades of software history to these environments. There is a belief that these environments have the potential of attracting new contributors to open-source projects. However, there is little empirical evidence to support these claims. In this paper, we quantitatively and qualitatively studied a curated set of open-source projects that made the move to GitHub, aiming at understanding whether and how this migration fostered collaboration. Our results suggest that although interaction in some projects increased after migrating to GitHub, the rise of contributions is not straightforward.**

I. Introduction

Along with the growth of open-source software (OSS), a wide range of social coding environments were created to support software development. These environments, which are particularly rich when it comes to collaboration features, changed the way that software developers communicate in, collaborate at, and contribute to OSS projects [1]. This is influenced by their contribution flow, the so-called pull-request (PR) model [2]. Developers clone (or "fork") the project, implement changes, and send the modifications back to the original project through a pull-request, which is evaluated by and discussed with project members.

Social coding environments helped to increase the number of contributors to OSS [2], [1], [3]. Gousios *et al.* [4] observed that in January 2016, 135,000 open-source projects hosted on GitHub, received more than 600,000 PRs. In addition, an increasing number of software developers are becoming open-source contributors [5], even though some of them do not wish to become active members [3]. This fact introduced the recurring belief that social coding environments foster engagement and collaboration within a project.

In this study, we investigate whether this growth in *popularity* can also be translated to *contributions*. Therefore, we selected a curated set of popular open-source projects that used to be hosted on non-collaborative coding environments (*e.g.,* Sourceforge) and moved to GitHub. Using data acquired from their repositories and a survey, we analyzed whether the migration process attracted more contributors and contributions. Among the collaborative coding environments, we focused on GitHub because of its popularity (it contained over 35 million repositories and more than 14 million contributors).[1] In addition, GitHub is often used in recent software engineering studies (*e.g.,* [3], [6], [7], [8]).

The main contributions of this paper are: (1) comparing quantitatively and qualitatively the contributions performed in popular OSS projects, before and after their migration to GitHub; (2) surveying project developers in order to cross-validate and further investigate the results; (3) making the dataset used in this study available.[2]

II. Related Work

Studying collaborative coding environments is an emerging direction. In this section we describe the studies overlapping with the scope of our work.

Github Social Features. Marlow *et al.* [9] found that developers use signals (skills, relationships, etc.) from the GitHub profile to form impressions of users and projects. Dabbish *et al.* [10] noticed that the number of watchers of a project serves as a social cue to attract developers. In sequence, Tsay *et al.* [8] found that both technical and social information influence the evaluation of contributions. McDonald and colleagues [5] found that the features provided by GitHub are cited as one of the main reasons to the increasing number of contributions and contributors to a project. These studies focused on collaborative coding environment features as drivers to attract developers and to generate signals to form impressions about projects and developers. However, they do not investigate how the migration to social coding environments influences the onboarding of new members and the number of contributions received by the projects.

Influence of Social Factors. Some studies focus on analyzing the influence of social factors in the retention of new developers [11], [12], [13]. These studies analyzed social networks (e.g., mailing lists) in order to understand (1) with whom new developers collaborate, and (2) how the networks evolve. Although these studies focus on the relationship of social aspects and onboarding of contributors, they do not analyze social coding environments as contribution drivers.

III. Research Methodology

In this section we state our research question (RQ) (§ III-A), the subjects under investigation (§ III-B), and the research approach (§ III-C).

[1] https://github.com/about/press

[2] https://github.com/fronchetti/ICSME-ERA-Dataset

978-1-5090-3806-0/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSME.2016.78

A. Research Question

We investigated the following overlooked research question:

RQ. How does the shift to GitHub impact projects' collaboration?

By projects' collaboration, we compared (1) the number of newcomers, active contributors, and contributions, (2) number of PRs received, and (3) number of issues created.

B. Subjects

For evaluation, we used a variety of software projects that migrated to GitHub: `jenkins`, `ruby`, `rails`, `jquery`, `mongodb`, and `joomla!`. Table II summarizes the characteristics of the studied projects. Some projects lack issues data because, although hosted on GitHub, they do not use GitHub's issue tracking system.

We selected these projects because they are: ***non-trivial*** (most of them with hundreds of thousands lines of code and use more than one programming language); ***well-established*** (with an average of 12 years old); ***active*** (they received an average number of 64 PRs per month and an average of 429 contributions in the last 12 months performed by 500 different contributors) and ***diverse*** (projects from different domains and written in different programming languages).

C. Research Approach

We conducted a two-step approach, investigating data and meta-data of the studied projects and questionnaires.

1) Mining Subjects: We used mining software repositories techniques to collect the following projects' characteristics:

- The number of newcomers that have joined the project over time, number of contributions (*i.e.*, commits) that have been performed to the project, the number of active contributors in a given time window.
- The number of PRs that are either *opened*, *closed*, or *merged*.
- The number of issues that are either *opened* or *closed*.

Next, we compared each collected metric per month before and after the migration of the studied subjects. To perform the comparisons, we used Mann-Whitney-Wilcoxon (MWW) tests [14] and Cliff's delta effect-size measures [15]. Both statistical tools are non-parametric, which means they do not require our collected metrics to follow a normal distribution. We use MWW tests to compare if two distributions do come from the same population (α = 0.05). We used Cliff's delta to verify how often values in one distribution are larger than values in another distribution. The higher the value of the Cliff's delta, the greater the difference between distributions. A positive Cliff's delta shows how larger are the values of the first distribution, whilst a negative Cliff's delta shows otherwise. We use the thresholds provided by Romano *et al.* [16]: $delta < 0.147$ (*negligible*), $delta < 0.33$ (*small*), $delta < 0.474$ (*medium*), and $delta >= 0.474$ (*large*).

2) Surveying Subjects: After collecting the data from the repositories, we designed our survey not only to cross-validate our findings from the repositories but also to gain further insights from the communities. Our survey is based on the recommendations of Kitchenham *et al.* [17]. Before sending the final survey, we conducted a pilot study and rephrased our questions whenever necessary.

Survey Questions. We designed the following three general open questions:

Q1 What motivated the project to move to Github? How do you evaluate the benefits of this migration?

Q2 Does this snapshot make sense? Did you find any inconsistency on the data?

Q3 Do you have any internal policy to promote/attract/retain newcomers? If so, do they succeed?

In addition to these questions, we also asked specific questions. The specific questions were aimed at revealing the reasons behind particular trends observed in each figure, for instance, why did the number of contributors decrease in a given time window, or why did the project attracted so much/so few external contributors? When relevant, we highlight them in Section IV.

Survey Application. We sent out the questionnaire by means of issues on the repositories. This approach worked as an effective feedback loop between the researchers and the respondents, making it easy to further clarify questions. For the repositories that do not use the issue tracking system, we sent the questionnaire by means of their official mailing lists.

During a period of over 30 days, we received 4 answers, although only 3 of them were analyzed. One was discarded because it did not provide any valuable insight. In total, we accumulate 16 messages from 11 participants. To compile the survey results, we qualitatively analyzed the answers following open-coding and axial-coding procedures [18].

IV. Research Results

We organize our findings in terms of Contributions and Contributors (§ IV-A), PRs (§ IV-B), and Issues (§ IV-C).

A. Contributors and Contributions

To provide an overview of our dataset, Figure 1 presents a temporal perspective of different characteristics. The dotted green line represents the number of newcomer contributors that successfully placed at least one *source code contribution* to the project repository. The dotted blue line represents the number of contributions performed during the software lifetime. The red line represents the number of active contributors in the particular time window. Contributions can be performed in terms of commits or PRs. The vertical dotted line indicates when the project migrated to GitHub. Finally, all of our obtained statistical results ($p - values$ and effect-size values) can be found in Table II.

Recruiting Newcomers. We observed that some projects attacked a significant number of contributors right after the project moved to GitHub, such as `rails` ($p - value = 0.001$

TABLE I

THE DIVERSITY OF OUR TARGET APPLICATIONS. LOC MEANS LINES OF CODE. MAIN PL MEANS MAIN PROGRAMMIN LANGUAGE. PR MEANS PULL-REQUESTS. AGE IS PRESENTED IN YEARS.

Projects	**Domain**	**Launched in**	**Migrated in**	**Main PL**	**LoC**	**Commiters**	**Commits**	**Issues**	**PR**	**Age**
`jenkins`	Continuous Integration Server	Nov. 2006	Nov. 2010	Java	191K	556	21K	—	2K	10
`ruby`	Programming language	Jan. 1998	Feb. 2010	C and Ruby	1,001K	95	40K	—	1K	18
`rails`	Web-application framework	Nov. 2004	Apr. 2008	Ruby	203K	3K	53K	8K	15K	12
`jquery`	JavaScript library	Mar. 2006	Apr. 2009	JavaScript	41K	263	5K	838	2K	10
`mongodb`	NoSQL database	Oct. 2007	Jan. 2009	C++	2,104K	324	31K	—	1K	9
`joomla!`	Content Management System	Aug. 2005	Sep. 2011	PHP	610K	726	26K	2K	8K	11

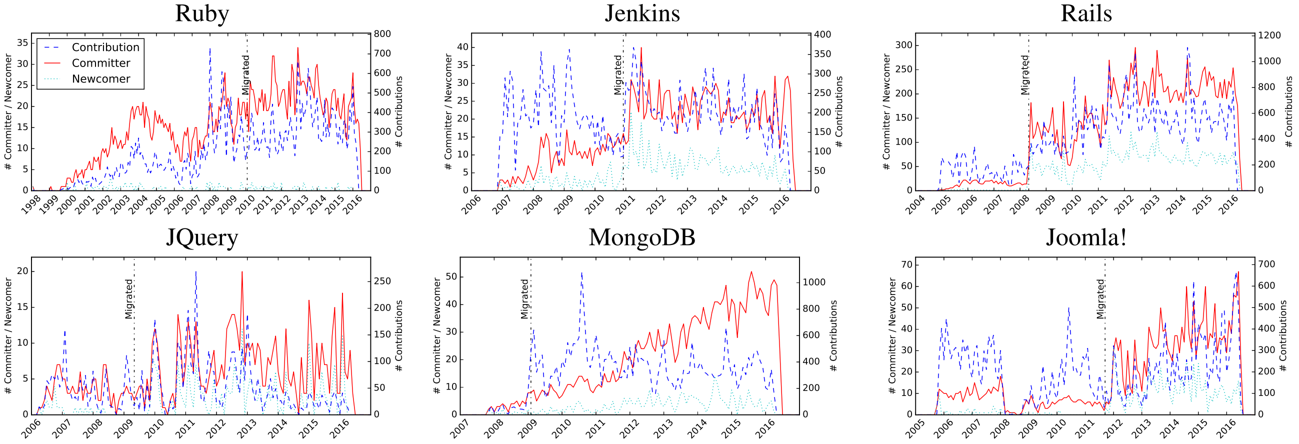

Fig. 1. **A temporal perspective of the number of contributions and contributors that on boarded in the project.** The vertical dashed line in each chart indicates when the studied project have migrated to github.

TABLE II

STATISTICAL RESULTS. GREEN CELLS INDICATE LARGE EFFECT SIZE, WHEREAS YELLOW CELLS INDICATE MEDIUM EFFECT SIZE.

Projects	**Newcomers**		**Contributors**		**Contributions**	
	p-value	*delta*	p-value	*delta*	p-value	*delta*
`jenkins`	0.001	0.131	2.66^{-06}	0.147	0.138	-0.031
`ruby`	0.489	-0.015	5.16^{-07}	0.106	2.20^{-16}	0.478
`rails`	0.001	0.460	2.20^{-16}	0.633	2.20^{-16}	0.635
`jquery`	0.060	0.160	0.001	0.144	0.035	0.066
`mongodb`	0.178	0.143	1.41^{-07}	0.403	2.20^{-16}	0.710
`joomla!`	0.006	0.206	2.20^{-16}	0.299	0.378	-0.018

with a *medium delta* $= 0.46$). In the `rails` project, the number of newcomers raised from 15 (before migration) to 67 (three months after migrating). On average, `rails` received 872 different newcomers per year. When asked why this happens, `rails` members suggested that one of the reasons are the GitHub's social features: "*It is easier to contribute. Traditional workflow of sending an email with the patch to an obscure mailing list while works, is not very user-friendly.*" One `jenkins` team member corroborates this finding: "*I think most projects that migrate there from other platforms obtain more contribution due to the basic popularity of GitHub and visibility.*"

Repelling Newcomers. On the other hand, other projects recruited fewer newcomers. For instance, `ruby`, in about two decades of software development, attracted only 96 different contributors (a rate of 5 newcomers per year). Since this particular project did not answer our survey, we hypothesize that this slow pace in attracting source code contributors might be related to the complexity of its internal code or domain.

Another reason is that the `ruby` programming language became popular around the 2000s, whereas GitHub was launched only in 2008 and became popular around 2011. Thus, the `ruby` project did not experience the benefits of GitHub's social features during its rise of popularity. Finally, the contribution guideline followed by `ruby` developers do not seem to facilitate contributions: "*Pull-request to https://github.com/ruby/ruby is acceptable for tiny fixes. But PRs which need discussions will be simply ignored.*"[3]

Dealing With Newcomers. In our survey, we asked if the project has any internal policy to deal with newcomers. However, none of the respondents mentioned such kind of policy, as one respondent said: "*Our team don't have any strong policy other than: be nice with the newcomers, give them the attention that they deserve, give them good feedback and try to motivate them to solve their own problems.*" A `jquery` respondent mentioned that, instead of an internal

[3] https://bugs.ruby-lang.org/projects/ruby/wiki/HowToContribute

policy, they use conferences and summits as means to attract new contributors: "*...developers summits and organic conversation in person have played the biggest roles in attracting newcomers*." Still, another team member raised the fact that in the end of 2012, when the `jquery` project faced a burst of newcomers, they held their first developer summit.

The migration to GitHub may have an impact on the rise of contributions. We could find evidence that the migration to GitHub is related to the increase of contributions of the majority of the studied projects. We observed that `rails`, `mongodb`, and `ruby` projects achieved *large* effect-sizes when we compare the number of contributions after the migration ($delta = 0.635$, $delta = 0.710$, and $delta = 0.478$, respectively). As for the number of contributors, we observed that `rails` and `mongodb` projects also achieved significant effect sizes after the migration (*large* difference with $delta = 0.633$ and *medium* difference with $delta = 0.403$, respectively). Nevertheless, we did not observe a substantial impact to the remaining projects. Overall, we observed that the migration to GitHub may play a major role to the rise of contributions in some of the studied projects (2 out of 5 in terms of contributors and 3 out of 5 in terms of contributions). However, there might be other project related factors (e.g., developer responsiveness and effective coordination) that may be hindering the increase of contributions in some projects. This was mentioned by one `jenkins` developer "*It's going to be hard to untangle what factor contributed how much.*" [to the amount of contributions received by the project].

B. Pull-Requests

As regarding PRs usage, Figure 2 shows the number of PRs opened, closed, and merged over time.

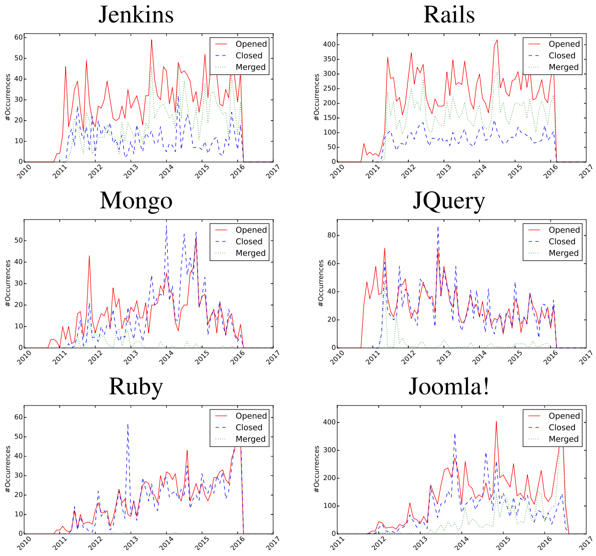

Fig. 2. Number of PRs opened (red lines), closed (blue dotted lines), and merged (green dotted lines).

The rise of PRs is not guaranteed. Since PR is usually the main channel to provide changes to GitHub hosted projects, one might expect that the number of PRs increases over time. However, our results suggest that this belief only holds for the `ruby` project. In this project, the number of PRs increased about 4 times from 2012 to 2016. In contrast, projects `jquery` and `mongodb` present a reduction on the overall number of PRs. This is, however, not related to a decrease in the overall number of contributions, as we saw in Figure 1. One reason that might explain this behavior is that core team members are not required to submit code changes through PRs; since they have write access to the repository, they can push changes directly to the main branch.

Several PRs are submitted, few got merged. Notwithstanding, as aforementioned, the `ruby` project uses PRs only for tiny fixes. As a result, we found that only 3 out of the 1,223 proposed PRs got merged. Projects `mongodb` and `jquery` present a similar behavior (9.86% and 8.20% of the PR got merged, respectively).

We manually investigated the `ruby` PRs that got merged and found that two of them were, indeed, tiny fixes (PR #557 and #194), but the third one (#125) was not (it has 379 additions and 9 deletions). However, the latter PR was performed by a ruby core developer, which usually does not need to follow the PR review cycle.

Steady flow of PRs. Although with variances, projects `jenkins` and `rails` present a steady flow of PRs contributions. On average, the `rails` project receive 2,072 PRs per year (variation: 1,478.73), whereas `jenkins` received 432.4 PRs per year, which a variation of 275.80. Also, these two projects present the highest rate of merged PRs, which might explain the steady flow of PR contributions, (*i.e.,* if core team members are accepting external contributions, it is likely that external contributors will keep contributing).

C. Issues

Figure 3 shows the number of issues created and closed during the lifetime of the analyzed projects. This figure shows the projects that use GitHub's issue tracking.

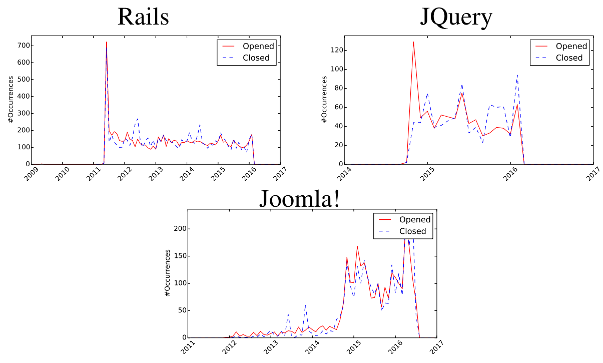

Fig. 3. Number of issues closed (blue dotted lines) and opened (red lines).

There is a peak in the number of issues right after the project start using GitHub. This happens with the `rails` and `jquery` projects. On average, the `rails` project has a rate of 136 new issues per month. In comparison, a total of 735 issues were created in the first month. A similar trend, but on a lower scale, is observed in `jquery`. We believe this happens because developers need to migrate their issues from other issue tracking system to GitHub. Yet, we hypothesize that not all projects are using GitHub issue tracking systems in order to avoid the initial effort required. On the other hand, `joomla!` presents a distinct scenario. During 2011 and 2014, there is a slow flow of issues. However, it peaks after mid-2014. Analyzing these issues, we found that in the second half is in fact related to the migration process from another issue track system, which corroborates to our hypothesis.

V. Discussions

Lessons Learned. First, we observed that the rise of contributions after the migration to GitHub is not straightforward. Indeed, one project faced a decrease in the flow of contributions. So the belief that GitHub itself will be effective in attracting new contributors to OSS projects does not capture the whole picture, although some project members agree that GitHub's features increase project visibility. Second, we found that little effort was placed on policies for attracting or retaining newcomers. Some projects use conferences and summits to attract new contributors. Third, we found that some projects have a high rate of PRs closed and not merged. Thus, if the project is not intended to use PRs, team members should state this upfront (*i.e.,* in the `README` file). Therefore, contributors willing to contribute will not spend their time providing PRs. Fourth, project members willing to move to GitHub might expect an overhead on managing issues migrated from other issue tracking system. Thus, project members should carefully consider this option.

Threats to Validity. One might argue that we analyzed too few open source projects and, therefore, it limits the generalization of our results. However, the selected open-source projects are diverse in terms of *domain*, *size*, and *age*. Another threat to validity is related to how we disambiguate commit authors. Since some of the analyzed projects moved from SVN-based environments, which do not distinguish author from committer, we used email address to disambiguate commit authors. However, SVN does not require one to inform his email. Also, one contributor might use different emails to perform different contributions. These facts have the potential of creating false-positive contributors, *i.e.,* the same contributor is counted more than once. To mitigate this threat, we used a disambiguate technique proposed by Bird *et al.* [19]. Also, we asked team members whether our data make sense. Finally, we used statistical methods to mitigate the threats of generalizing data based on our personal hypothesis.

VI. Conclusions

Social coding environments are changing the way software is built. These environments leverage social features that make contributions to software much more visible. Along with their popularity and these key features, these environments are being responsible for attracting new contributors to open-source projects hosted on them. In this paper, we studied whether this belief holds true for the analyzed projects. Among the results of our study, we found that although some projects increased interaction after migrating to GitHub, the rise of contributions is not guaranteed.

Acknowledgment

We thank the anonymous reviewers for their useful comments. This work is sponsored by FAPESP (2014/21899-4), NAPSoL, CNPq (477831/2013-3), and PROPESQ/IFPA.

References

[1] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider, "Creating a shared understanding of testing culture on a social coding site," in *ICSE*, 2013, pp. 112–121.

[2] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *ICSE*, 2014, pp. 345–355.

[3] G. Pinto, I. Steinmacher, and M. Gerosa, "More common than you think: An in-depth study of casual contributors," in *SANER*, 2016, pp. 112–123.

[4] G. Gousios and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *ICSE*, 2016, pp. 358–368.

[5] N. McDonald and S. Goggins, "Performance and participation in open source software on github," in *CHI*, 2013, pp. 139–144.

[6] I. Moura, G. Pinto, F. Ebert, and F. Castor, "Mining energy-aware commits," in *MSR*, May 2015, pp. 56–67.

[7] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *FSE*, 2014, pp. 155–165.

[8] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *ICSE*, 2014, pp. 356–366.

[9] J. Marlow, L. Dabbish, and J. Herbsleb, "Impression formation in online peer production: Activity traces and personal profiles in github," in *CSCW*, 2013, pp. 117–128.

[10] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *CSCW*. New York, NY, USA: ACM, 2012, pp. 1277–1286.

[11] M. Zhou and A. Mockus, "Who will stay in the floss community? modelling participant's initial behaviour," *IEEE Transactions on Software Engineering*, vol. 41, no. 1, pp. 82–99, 2015.

[12] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *CSCW*, vol. 14, no. 4, pp. 323–368, Aug. 2005.

[13] C. Bird, "Sociotechnical coordination and collaboration in open source software," in *ICSM*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 568–573.

[14] D. Wilks, *Statistical Methods in the Atmospheric Sciences*, ser. Academic Press. Academic Press, 2011. [Online]. Available: https://books.google.com.br/books?id=IJuCVtQ0ySIC

[15] R. Grissom and J. Kim, *Effect Sizes for Research: Univariate and Multivariate Applications*. Taylor & Francis, 2005.

[16] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys?" in *Annual meeting of the Florida Association of Institutional Research*, 2006.

[17] B. Kitchenham and S. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. Sjberg, Eds. Springer London, 2008, pp. 63–92.

[18] A. Strauss and J. M. Corbin, *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory*, 3rd ed. SAGE Publications, 2007.

[19] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 137–143.