Analyzing the Impact of Social Attributes on Commit Integration Success

Mauricio Soto, Zack Coker, and Claire Le Goues
School of Computer Science
Carnegie Mellon University, Pittsburgh PA
mauriciosoto@cmu.edu, zfc@cs.cmu.edu, clegoues@cs.cmu.edu

***Abstract*—As the software development community makes it easier to contribute to open source projects, the number of commits and pull requests keep increasing. However, this exciting growth renders it more difficult to only accept quality contributions. Recent research has found that both technical and social factors predict the success of project contributions on GitHub. We take this question a step further, focusing on predicting continuous integration build success based on technical and social factors involved in a commit. Specifically, we investigated if social factors (such as being a core member of the development team, having a large number of followers, or contributing a large number of commits) improve predictions of build success. We found that social factors cause a noticeable increase in predictive power (12%), core team members are more likely to pass the build tests (10%), and users with 1000 or more followers are more likely to pass the build tests (10%).**

***Keywords*-Social Attributes; Travis CI; GitHub; Social Networks; Predicting Integration Success; Social Coding**

I. Introduction

For managers of an open source project, it is important to ensure that the project consists of high quality contributions. As social coding sites attract more users and projects become more popular, they receive ever more community contributions. As the number of project contributions increases, the project must dedicate more time to vetting these contributions. The open source community has developed various techniques to vet project contributions, such as requiring that pull requests be approved by a core team member [4]. This can become a bottleneck, and thus it is beneficial to prioritize certain contributions, ideally by predicted quality.

Ideally, the decision to include code in a project is meritocratic, such that only code quality is a factor in the decision [11]. However, beyond commit complexity or correctness, past studies have shown that the social factors describing the committing developer also importantly predict contribution success [12], [13]. Thus, both technical factors, which describe the commit's complexity, and social factors, which describe the developer's experience, are important when investigating contribution success (and thus by extension predicted quality).

This previous work has focused on the predicted acceptance of open source *pull requests* [12], [13] and *continuous integration* builds in a single project [7], [8], [15]. By contrast, we are interested in analyzing contribution quality in *continuous integration* across multiple open source projects. While pull request quality is ultimately determined through manual inspection, continuous integration uses test cases to automatically assess commit quality, and thus may provide different quality predictors. We investigate the factors related to continuous integration build success across open source projects. In this study, we analyze a large set of technical and social attributes and how these can be used to predict a contribution's quality, measured by the success of build integration jobs. We performed this analysis on a large corpus of open source projects which incorporated continuous integration in their development practice. When a contribution to these projects passes integration tests, it suggests that the project developers might have confidence in the changes. Thus, we use the commit's build integration status as a proxy for its quality.

We analyzed contributions in the context of continuous integration using technical factors and one social factor from the challenge corpus [2], and social factors from a developer's GitHub profile. We used the technical factors to estimate the commit's complexity, and the social factors to estimate the developer's experience. Overall, our study investigates how different technical and social factors predict contribution quality. This information can help core team members prioritize project contributions, and increases understanding of quality enforcement in software development.

We find that social factors are an important predictor of build success, but do not fully predict whether a contribution will pass all build tests; that core team members are more likely to achieve build success than non-core members; and that developers with more followers are more likely to submit commits that pass the continuous integration build.

II. Research Design

We present the approach we took to design our study by following the Goal-Question-Metric (GQM) methodology [1]:

Goal. The *goal* of this study is to understand the relationship between social factors that describe developer's experience (stars, followers, etc.) and technical properties that describe a commit's complexity (files added, files modified, etc.) when predicting whether a developer's commit will pass the continuous integration build tests for the project.

Question. We are interested in understanding whether social factors, used as a proxy for experience, can be used to predict commit integration success. If so, developers and integrators

will have another tool to estimate the quality of a developer's contribution. Our main research question is:

> *Do social attributes improve prediction of a contribution's build success when compared to predictions based only on technical attributes?*

If social attributes improve build success predication, we are interested in investigating how particular attributes, such as the developer's follower count, similarly correlate with success.

Metric. We measured software contribution quality by the build status of the commit. We used a decision tree classifier [9] to predict build success. Decision trees use binary decisions on the attributes to predict a final class. They are well established and interpretable. We used a Weka J48 decision tree for readability. We first built a decision tree classifier using only technical factors to predict build integration success. We then included the social factors, to identify the degree to which social factors improve the model's predictive power.

III. Building the Dataset

We built a data set consisting of both technical and social factors:

Technical Factors. We collected technical factors from the Mining Software Repositories (MSR) Challenge data set [2]. We extracted the following technical attributes to proxy a commit's complexity:

- Source churn, changed lines of code.
- Files added by this commit.
- Files modified by this commit.
- Tests added by this commit.
- Tests deleted by this commit.
- Total source files in project at the time of the commit.
- Number of source lines of code in the project at the time of the commit.
- Travis status, or status of the Travis build integration.

The first five factors describe the changes in the commit. The next two factors cover the size of the project. These factors proxy both the commit's complexity, and the complexity of the project incorporating it.

Social Factors. We created a web scraper to extract social data from developers' profiles. We used the commit hash and the project name in the MSR Challenge [2] data set to locate the commit web page, leveraging *github.com*'s URL standard for projects and commit hashes. We scraped the commit author's *username* from each commit page to collect the following from the user's public profile:

- Number of repositories the developer can access (including those the developer owns, contributes to, or for which they have organizational membership access).[1]
- Number of projects the developer has stared. Starring a project to shows approval of the repository and creates a bookmark for later access.[2]
- Number of developers that follow the author.
- Number of developers that the author follows.
- Number of contributions made in the last year.

We also added one item from the MSR Challenge data set, indicating whether the commit's developer is a core member of the associated project. These social factors are all tied to a GitHub user's profile, and are generally correlated with users' open source engagement. Thus, we use them as a proxy for experience. If we were unable to find information about an author, we assigned empty values to the commit.

Compiling the data sets. We compiled the gathered data into two data sets: one with only the technical factors and the build status, and another with both technical and social factors and the build status. We performed this split to analyze how well technical factors alone, and then both technical and social factors together predict build integration status.

The challenge data set [2] consists of commits that date as far back as 2011. The social data we gathered is accurate for January 2017. To mitigate the threat of using future results to predict previous events, we only include commits from the years 2014–2016. Our intuition is that these social attributes are unlikely to have significant recent changes.

We converted all Travis build statuses to two possible values: *successful* (passed in the challenge data set) and *unsuccessful* (errored, failed, canceled or started in the original data set). We also condensed all passing and failing builds in a commit to a single pass or fail for each build status. If the commit contained both passing and failing build statuses, we treat it as both a passing and failed build.

74.43% of the builds in the reduced data set were successful, and thus a naive classifier that always guesses "successful" would succeed 74.43% of the time. To penalize misclassification for both classes equally, we created a second *balanced* data set by randomly sub-sampling the successful builds until the two build status were equal, through a process known as downsampling. We downsampled the entire data set and then trained decision tree classifiers on both. We validated the classifiers with cross-validation. Since the cross-validation is stratified, the 50-50 proportion is maintained through the training and testing folds for the downsampled data. We present results for both data sets in Section IV-A, and only the full data set in Sections IV-B and IV-C.

IV. Evaluation

We assess our main research question using decision tree classifiers [9], using the Weka J48 decision tree [6], [10] with default settings and 10-fold cross-validation. We present classification success rate for technical versus social factors (Section IV-A); build success rate of core project members versus non-core members (Section IV-B); and the relationship between follower count and build success (Section IV-C).

A. Classification with and without social factors

Table I shows the results of running the two data sets through the default Weka J48 decision tree algorithm. Without downsampling, the decision tree shows an increase of 4.09% *correctly classified* instances when adding social data. On

[1] https://developer.github.com/v3/repos/

[2] https://help.github.com/articles/about-stars/

	Non-Downsampled		Downsampled	
	Technical	Technical and Social	Technical	Technical and Social
Correctly Classified Instances	285882 (77.01%)	301068 (81.10%)	121694 (64.10%)	144441 (76.09%)
Incorrectly Classified Instances	85326 (22.99%)	70140 (18.90%)	68147 (35.90%)	45400 (23.91%)
Kappa statistic	0.21	0.42	0.28	0.52
Average Precision	0.75	0.80	0.65	0.76
Average Recall	0.77	0.81	0.64	0.76
Average F-Measure	0.72	0.79	0.63	0.76
Mean absolute error	0.34	0.28	0.43	0.32
Increase when adding social attributes	4.09%		11.99%	

TABLE I
PREDICTIVE POWER DECISION TREES WITH A NON-DOWNSAMPLED DATA SET (2ND AND 3RD COLUMN) AND DOWNSAMPLED (4TH AND 5TH COLUMN).

the downsampled data, the decision tree created from the technical attributes had a classification accuracy of 64.10%, which outperformed the naïve baseline model that guesses all builds passed (50.00%). After including the social data in the data set, the classifier correctly predicted 76.09% of the data set, about a 12% improvement over the model that uses only the technical data. These results show that while technical data is helpful in predicting build integration success, social data can be used to measurably improve predictive accuracy.

The precision, recall, and F-measure are calculated over the successful and unsuccessful classes and then averaged (standard in Weka). All these measures, along with the Kappa statistic, increase with social features, which provide evidence that social factors improve build success predictions. The mean absolute error also decreases, further substantiating the result.

B. Build success by core team member

The decision tree results suggested that core team membership is important to classifying build success (in supplemental material), and thus we investigated the question directly. We classified the build success rate for core team members and non-core team members using the data set without downsampling. We found that core team members triggered 89.39% (331827/371208) of the builds in the data set; 75.58% (250808/331827) of those builds passed. Non-core team members triggered 10.61% (39381/371208) of the builds in the data set, and 64.70% (25479/39381) of the builds passed. The difference in success rate is statistically significant (the p-value from a Fisher exact test is 0.0000), suggesting that it is extremely unlikely to have occurred by chance. Thus, in our dataset, core team members are more likely to have a successful build integration than non-core team developers, likely because core team members are naturally more familiar with the codebase.

C. Follower relationship with build success

We also investigated whether developers with followers were more likely to have a successful build integration, since followers could be seen as "vouching" for a developer's abilities. We therefore investigated the relationship between follower count and build success rate (on the data without downsampling). We first investigated whether developers with one or more followers were more likely to pass the build than those without any followers at all. We removed commits without any author profile information (20374 builds, reducing the total build count to 350834). Only 0.87% (3059/350834) of the contributions in the data set were performed by developers with zero followers. Developers with 0 followers had a successful build commit 68.49% (2095/3059) of the time. Developers with one or more followers were successful 74.78% (260056/347775) of the time. The p-value of this difference, calculated with a Fisher exact test, is 0.000, suggesting the difference in commit success rate between developers with no followers and those with at least one follower is very unlikely to have occurred by chance. By contrast, developers with a large number of followers (1000 or more), performed 6.94% (24347/350834) of the contributions in the data set; their contributions had a 78.61% (19138/24347) success rate. The p-value of the difference between this success rate and that of developers with no followers is again 0.0000.

These results suggest that developers with more followers are more likely to have a successful integration build. This may be due to the fact that developers with followers are more likely to have open source development experience, and the developers' experience influenced their build success.

V. THREATS TO VALIDITY

In the study, we use Weka, a well known machine-learning toolkit, to analyze our data. Like all tools, Weka may contain bugs. Nonetheless, it is a community tested open source tool, which mitigates threats to internal validity. We also release our scripts, data, and decision tree results in the study.[3]

We mitigate the risk that our results fail to generalize externally by studying a broad and rich data set [2] comprising 1,359 popular projects. Similarly, the social factors we study may not apply to closed-source applications; however, open-source ecosystems are intellectually valuable sources of study on their own. Other concerns include the possibility that social factors have changed significantly in the past three years, that unsuccessful builds are due to incorrect test cases (rather than low quality contributions), and that information is lost in collapsing commits' multiple builds. We believe these are similarly mitigated by a large and varied data set, but leave investigation of the effects of these decisions to future work.

[3]https://github.com/squaresLab/MSR-challenge-2017

VI. Related work

The most similar related work to this paper is a case study investigating how social and technical factors affect build success on a project with 40 developers [7]. We focus on a wide range of open source projects and measure experience through developer profile information. Other work has investigated the effects of various factors, such as team organization, on predicting build success in a single project [8], [15]. Previous investigations have established relationships between both social and technical attributes and different measures of success, such as pull request acceptance [4], [5], [12], [13] or evaluation latency [17]. Tsay et al. [12] show the importance of the social connection between the developer initiating a pull request and the project integrators. We look at a broader aspect of social factors, and focus on predicting the Travis CI build success. Gousios et al. [4] found that the commit change location is an important indicator to commit acceptance rate and that including test cases increases a commit's acceptance rate [5]. We found that number of followers and being a core member are important indicators of build success. Bettenburgh et al. [3] discuss the impact of social interactions with post-release defects, while Vasilescu et al. [14] explore how different project characteristics (programming language, project age, etc.) affect commit success when integrated into the automatic build process. In our study, we focus on the comparison between similar technical characteristics and social attributes. Yu et al. [16] analyze the impact of CI usage in GitHub and its correlation with software quality, while we focus on gain in predictability of software quality when using CI.

VII. Conclusion

We have analyzed how technical and social factors can be used to predict high quality open source contributions, measured by continuous integration build success. We found social factors improve predictions of build success; technical factors led to a 14% increase in predictive power, and adding social factors led to another 12% increase.

Overall, we conclude that social attributes proxying experience, such as whether the developer is a core project contributor, noticeably influence build success prediction, especially on the downsampled data. In particular, core team members are more likely to have the build integration pass than non-core team members. This result provides evidence that non-core team member commits may not be scrutinized as closely as core team member commits before the integration build tests, although further study is required to draw definitive conclusions. Likewise, developers with more followers are more likely to pass the build. Developers with 1000+ followers were more than 10% more succesful in their builds compared to developers with no followers. This suggests that open source developers with high quality commits may be more likely to have a large number of followers.

Although we find experience is an imperfect indicator, experienced developers appear more likely to make high quality commits.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1252522. The authors also gratefully acknowledge the partial support of the AFRL (AFRL Contract No. FA8750-15-2-0075). Any findings or recommendations are those of the author(s) and do not necessarily reflect the views of the sponsoring agencies.

References

[1] Victor R Basili and David M. Weiss. A methodology for collecting valid software engineering data. In *Transactions on Software Engineering*, pages 728–738, 1984.

[2] Moritz Beller, Georgios Gousios, and Andy Zaidman. TravisTorrent: Synthesizing Travis CI and GitHub for full-stack research on continuous integration. In *Mining Software Repositories*, MSR '17, page (to appear), 2017.

[3] Nicolas Bettenburg and Ahmed E. Hassan. Studying the impact of social interactions on software quality. In *International Conference on Program Comprehension*, ICPC '10, pages 124–133, 2010.

[4] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *International Conference on Software Engineering*, ICSE '14, pages 345–355, 2014.

[5] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. Work practices and challenges in pull-based development: The integrators perspective. In *International Conference on Software Engineering*, ICSE '15, pages 358–368, 2015.

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, November 2009.

[7] Ahmed E. Hassan and Ken Zhang. Using decision trees to predict the certification result of a build. In *Automated Software Engineering*, ASE '06, pages 189–198, 2006.

[8] Irwin Kwan, Adrian Schroter, and Daniela Damian. Does socio-technical congruence have an effect on software build success? A study of coordination in a software project. *Transactions on Software Engineering*, 37(3):307–324, May 2011.

[9] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[10] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[11] Walt Scacchi. Free/Open source software development: Recent research results and emerging opportunities. In *Foundations of Software Engineering: Companion Papers*, FSE '07, pages 459–468, 2007.

[12] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *International Conference on Software Engineering*, ICSE '14, pages 356–366, 2014.

[13] Jason Tsay, Laura Dabbish, and James Herbsleb. Lets talk about it: Evaluating contributions through discussion in GitHub. In *Foundations of Software Engineering*, ICSE '14, pages 144–154, 2014.

[14] Bogdan Vasilescu, Stef van Schuylenburg, Jules Wulms, Alexander Serebrenik, and Mark G. J. van den Brand. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *International Conference on Software Maintenance and Evolution*, ICSME '14, pages 401–405, 2014.

[15] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *International Conference on Software Engineering*, ICSE '09, pages 1–11, 2009.

[16] Yue Yu, Bogdan Vasilescu, Huaimin Wang, Vladimir Filkov, and Premkumar Devanbu. Initial and eventual software quality relating to continuous integration in GitHub. *Computing Research Repository*, http://arxiv.org/abs/1606.00521, 2016.

[17] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: Determinants of pull request evaluation latency on GitHub. In *Mining Software Repositories*, MSR '15, pages 367–371, 2015.