Co-evolution of Project Documentation and Popularity within Github

Karan Aggarwal, Abram Hindle, Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
{kaggarwa,hindle1,stroulia}@cs.ualberta.ca

ABSTRACT

Github is a very popular collaborative software-development platform that provides typical source-code management and issue tracking features augmented by strong social-networking features such as *following* developers and *watching* projects. These features help "spread the word" about individuals and projects, building the reputation of the former and increasing the popularity of the latter. In this paper, we investigate the relation between project popularity and regular, consistent documentation updates. We found strong indicators that consistently popular projects exhibited consistent documentation effort and that this effort tended to attract more documentation collaborators. We also found that frameworks required more documentation effort than libraries to achieve similar adoption success, especially in the initial phase.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics— *Process metrics*

General Terms

Documentation, Human factors

Keywords

Documentation Change, Popularity, Cross Correlation

1. INTRODUCTION

Recently, we have been witnessing the emergence and increased adoption of on-line collaborative software-development platforms, such as SourceForge, GitHub, and Google Code, to mention a few. GitHub, based on the Git version-control system, affords software-development support (including source control and issue tracking) but also social interactions. For example, in addition to *forking* code repositories and merging local branches to the master branch with *pull requests*, GitHub members can *follow* other members and subscribe to project repositories to *watch/star gaze* them. These features distinguish GitHub from conventional repository-hosting platforms, in that they enable a much richer set of interactions among the community members that has a pronounced effect on the repositories [1].

In this paper, we study the effect of these social interactions on project documentation. We examined the projects within the MSR'14 data-set [3], which is a representative data-set of projects from different programming languages, to answer following questions:

RQ1: How does the popularity of a project correlate with its documentation?

RQ2: Do different types of projects exhibit different documentation-evolution behaviours?

In the rest of our paper, we place our work in the context of related studies (Section 2); describe our methodology (Section 3) and findings (Section 4) and we conclude with a summary and some ideas for future work (Section 5).

2. RELATED WORKS

To the best of our knowledge, the interplay between project popularity and software documentation has not yet been studied on social development platforms like Github. However, there has been substantial interest in how social features enhance collaborative software development in general.

Early on, Storey *et al.*[6] suggested a number of research questions on interactions between the software development process and the social aspects offered by these platforms, including the interplay between traditional project documentation and informal information sharing through social media.

Lee *et al.*[4] discussed the effect of `Rockstar` users to the project popularity. `Rockstar` users have a large number of followers and are thought of as being particularly skilled; their reputation may possibly attract more attention to their projects. The study found that *pull requests* by `Rockstar` developers induces follower activity. Hence, *pull requests* can be through of as an important indicator of project popularity.

Dabbish *et al.*[1] explored the effects of transparency in GitHub, as a result of *following* and *watching* activities. They surveyed a group of 24 Github developers with different motivations of participation on Github. They found that developers make lot of inferences about projects from the social activity, like liveliness by the commit activity; the owner's activeness in the way the pull requests are handled; and most importantly, the project's popularity by the fork and watcher count.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
ACM 978-1-4503-2863-0/14/05
http://dx.doi.org/10.1145/2597073.2597120

Dagenais et.al.[2] conducted a study on the evolution of the project documentation, by surveying 22 developers working on a variety of open-source projects, including applications, libraries, and/or frameworks. They found that documentation evolves in three steps: initial effort, incremental changes, and change bursts. For library projects, the initial documentation phase was difficult, and the documentation evolved as the requirements became clearer and the library became more popular. Framework projects required more initial documentation effort (as frameworks require their users to compose many parts together). Irrespective of the project type, the initial effort invested in documentation plays an important role in project popularity, especially in attracting users from competing projects.

Shi *et al.*[5], studied the API documentation evolution, concluding that API documentation change is quite different across the versions with documentation change being proportional to the difference between version numbers.

In this paper, we look at the evolution with respect to time, as opposed to the release versions. We use the terms repository and project interchangeably in this paper.

3. METHODOLOGY

In this section, we first describe the metrics we defined as measures of documentation evolution and project popularity; next, discuss our data-collection and analysis methodology.

3.1 Metrics

We define the Change and Popularity metrics to empirically analyze our questions defined in Section 1. These metrics are **defined per month**.

Documentation Change.

We assume that documentation, considering only non-code documentation, is contained within these file types (matched by extension): *pdf, txt, md, jpg , png, ps* and *mp4*(most of *mp4*'s were tutorials).

In devising a documentation-change metric, we had to take into account not only the size of the change (number of lines changed) but also the frequency of changes, as an indication of the contributor activity, and how central documentation maintenance is considered in the project. We define the documentation-change (per month) metric as:

$$Change = N\ log(\sum_{i=1}^{N} c_i + 1) \quad (1)$$

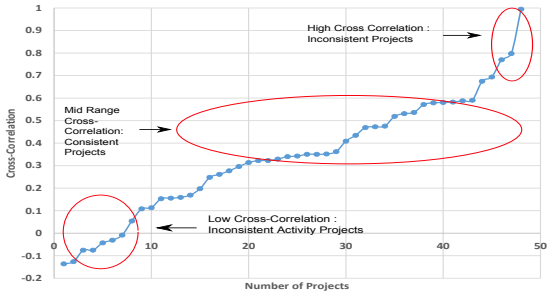


Figure 1: Vertical axis shows the cross-correlation between the popularity and documentation change while the horizontal axis shows the number of projects

Here, N is the total number of files changed, c_i is the net change per file, i.e. number of lines added minus the number of lines deleted from the file. We choose net change because of the way Github counts the deletions and additions; even reordering the phrases is counted as one addition and one deletion, and skews the data. We took the log to make the value range comparable to the popularity metric and multiplied by N to take into effect of frequency of change.

Popularity.

Inspired by the work of Dabbish *et al.*[1] who found that developers perceive number of forks and watchers as an indicator of project popularity, we defined project popularity as follows:

$$Popularity = (\#Stars) + (\#Forks) + (\#Pulls)^2 \quad (2)$$

We also added a term for the pull requests as since Dabbish *et al.* [1] and Lee *et al.* [4] conclude that pull requests are an important indicator of the activity and popularity of a project.

3.2 The Data Set

Our data set is based on the MSR'14 Mining Challenge data set [3] which has 90 projects extracted from Github. We obtained the *commits*, *forks*, *watchers* and *pull requests* for these selected 90 projects. We obtained the individual file commits with commit timestamps, and number of lines added and deleted, resulting in about 5 million entries. We also collected 76 000 forks and 300 000 watchers.

On this data-set, we first calculated the documentation-change metrics per month per project while filtering log files. Next, we calculated the project code size for every month by considering the code files. We control against size in our linear regression analysis in Section 4.1. Next, we calculated the project popularity per month. Finally, we constructed a super joined data-set containing the monthly popularity, change and size for each project. Project maturity varied between 2 months to 10 years.

3.3 Approach

To argue that documentation matters with respect to popularity we have to show their relationship. We chose to correlate the time-series of popularity with the time series of documentation change. If popularity matters, then after an increase in popularity we should see a consistent documentation change. Thus in order to evaluate RQ1,that is to establish the similarity between the two time series,we use cross correlation - the correlation with lag between two time-series, and is defined as:

$$\rho_{cross} = \arg\max_{\tau} \rho(Change_{t+\tau}, Popularity_t) \quad (3)$$

where, τ is the time lag for the popularity to have its effect onto documentation which might vary across a project's lifetime. Furthermore, it might be affected by external factors such as blog posts or marketing. ρ_{cross} is maximum correlation value obtained from $1 \leq \tau \leq 11$.

When evaluating RQ1 we consider only those projects which are at least one year old, and where most tuples of $(Change_{t+\tau}, Popularity_t)$ are not zero (0,0). If we do not filter out projects with many zero measurements then due to ties we will get very high correlations. Initially, most projects do not have much documentation change or popularity. This leaves us with 48 projects out of 90.

4. ANALYSES AND FINDINGS

This section discusses the findings of our exploration of the research questions identified in Section 1 using the methodology discussed in the previous section.

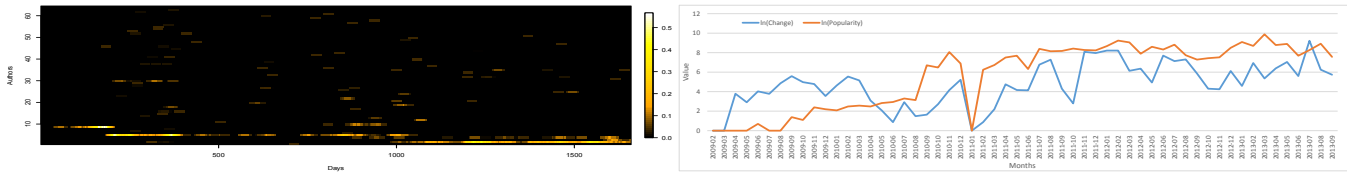
4.1 RQ1:How does project popularity correlate with project documentation?

We use cross-correlation analysis, as discussed in Section 3.3, on the resultant data-set of 48 projects. As we can see in Figure 1, most projects show a positive correlation. Out of 48 projects, 7 projects had negative correlation, while 12 projects show correlation between 0 and 0.3; the remaining 29 projects have correlations greater than 0.3.

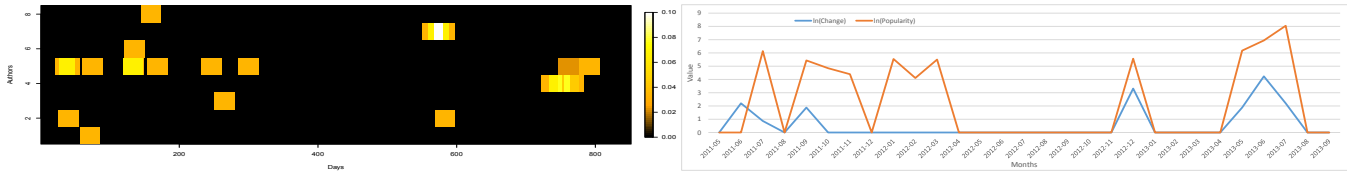
In order to control for size, we used the project size in our linear regression model. Popularity and size were independent variables regressing on the dependent variable of the future documentation change. We concluded that size does not have any effect on these two variables as it had negligible coefficients with insignificant p-values > 0.05 for 75% of the projects. 35% of the project had models where popularity was significant, but for more than 80% of the project models the sign of popularity's coefficient was stable. The reverse model of change inducing popularity had even less significant results and less coefficient stability. This provides evidence that popularity induces documentation change for some projects.

Careful inspection of the negative-correlation projects revealed cases such as `mbostock/d3` that had many pull requests pending from as long as 2 years and issues pending for 3 years. High positive correlation projects, except for `facebook/folly` which had correlation near 1, tended to have a good track record of resolving issues and merging the pull requests. `folly` had high correlation because it had a point where popularity and changes peaked, after that the project had very sporadic activity and popularity.

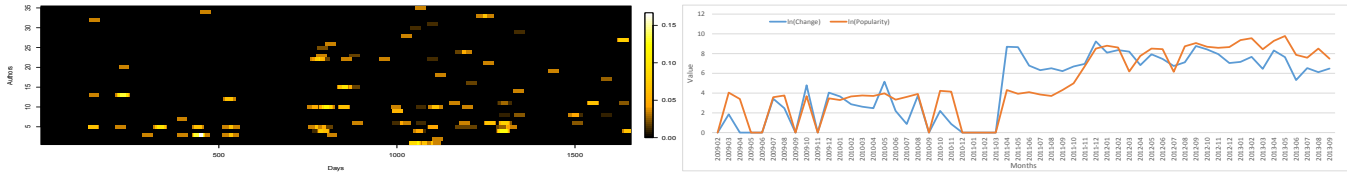
This observation led us to divide the projects in two categories: those having a *sporadic popularity* and those with a *consistent popularity*. The former category includes project whose popularity metric has occasional bursts of high values for a short time but is zero for the major part of the project life-cycle. We manually categorized projects into



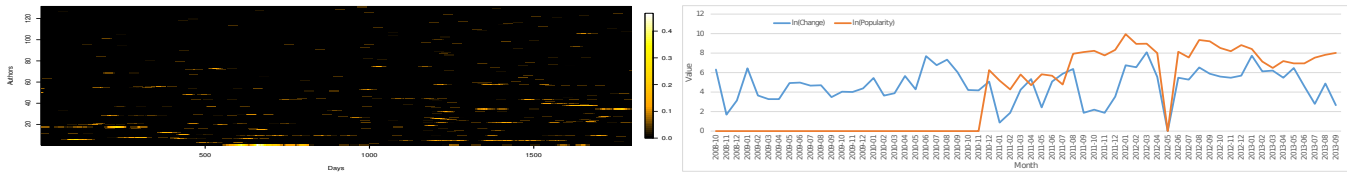
(a) Documentation Activity (Left) and Change vs Popularity graph(Right) for `Node`



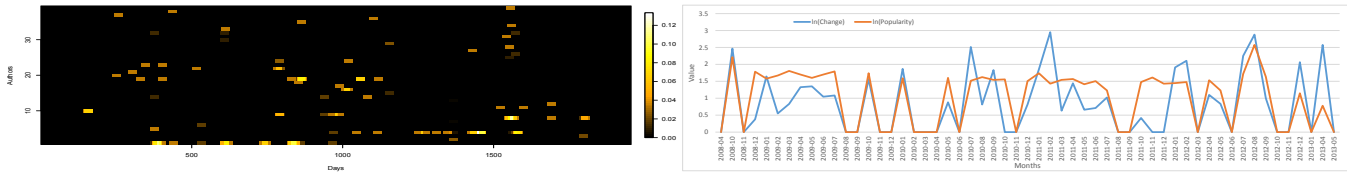
(b) Documentation Activity (Left) and Change vs Popularity graph(Right) for `Chosen`



(c) Documentation Activity (Left) and Change vs Popularity graph(Right) for `akka`



(d) Documentation Activity (Left) and Change vs Popularity graph(Right) for `TrinityCore`



(e) Documentation Activity (Left) and Change vs Popularity graph(Right) for `MaNGOS`

Figure 2: Left Plot shows the commit average documentation commit activity per day (over a window of 30 days) per each author on the y-axis while Right Plot shows the ln(Change) vs. ln(Popularity) with respect to project timeline on the x-axis

the consistency and sporadically popular categories and we found that the projects having the sporadic popularity were at the bottom of the cross-correlation graph, whereas the projects having consistent popularity had positive cross-correlation. However, there were notable exceptions, like `facebook/folly`, which had inconsistent activity but it features at the top of the graph. Out of 11 projects identified as sporadically popular, we find 7 at the bottom, 1 in the middle and 3 at the top range as identified in Figure 1. The three at the top exhibit a similar behaviour to `folly`.

Furthermore, the collaborators who were active in the documentation are quite different in the two groups. The consistent projects tend to have more collaborators who had some contribution to the documentation. For example, to show the disparity, let's consider two projects: `mbostock/d3` (sporadic activity) and `joyent/node` (consistent activity), with cross-correlations of -0.13 and 0.69, respectively. While `d3` has only one contributor to the documentation, spread across 10 distinct days of a documentation commits, `node` has 64 distinct collaborators having at least one documentation commit, spread across 370 days. Both projects have a similar number of forks and watchers of around 25000, as well as similar non-documentation commit activity. We show in Figure 2a and 2b, the contrast in the activity of these projects: `choosen` and `node` respectively, having correlation of -0.075 and 0.69. We can see `choosen` has less activity and fewer collaborators than `node`.

From the above analysis, we can see that there is a clear relationship between the popularity and the documentation effort: consistently popular projects exhibit higher (and more consistent) documentation activities and efforts.

4.2 RQ2: Do different project types differ in their documentation?

Intrigued by the conclusions of Dagenais *et al.* [2], we examined if and how documentation behaviours differ across different project types, i.e., frameworks and libraries. Studying the materials available on the project web sites, we manually categorized the projects in our data set in three types: 15 projects were clearly identified as frameworks, 12 as libraries and rest, 21 as neither. Frameworks had a mean cross-correlation of 0.35, libraries 0.352 and the third group 0.29.

Let us consider two specific projects to see the difference between libraries and frameworks, if any — `akka` and `TrinityCore`: the former is a library/API project and the latter is a framework project. `akka` and `TrinityCore` were chosen for their maturity of greater than 4 years. We found that most library projects followed a pattern similar to Figure 2c right graph, showing the popularity vs. documentation change over time for the `akka` project. As can be seen from the graph, there is very small initial documentation addition, that starts growing as the project's popularity shows a consistent uptick after the project life-cycle midpoint. It is quite interesting to see that both lines trace each other with a lot of similarity. Similarly, we found that most framework projects followed a pattern similar to Figure 2d right graph for the `TrinityCore` project. As we can see, there is already a lot of documentation before the popularity of the repository increases. This is required in order to document and compose the various libraries used by the framework. The graph for the third category is shown in 2e, for the `Mangos` project. The graph is included simply for illustrative purposes in comparison to the other two, as there is no consistent pattern in this group.

Hence, we can say that there is an strong indication that library projects require less initial documentation than the frameworks as suggested by Dagenais *et al.* [2].

4.3 Threats to Validity

There are two types of threats to the validity of our findings. Internal validity is threatened by the choice of month as the unit of time, whereas it was possible to have smaller measurement units like weeks. In addition, the selection of projects that have at least 50 percent non-zero data points could possibly have led to filtering out projects that were important. Our change and popularity metrics could affect the construct validity of the study. The external validity of our study is threatened by the choice of the projects in the MSR'14 data-set. We could mitigate this by considering more projects that have not been covered in the data-set.

5. CONCLUSIONS AND FUTURE WORK

In this study, we investigate the relationship between project popularity and changes in its documentation. Based on the results reported in Section 4.1 to answer RQ1, we can conclude that popularity does help in the evolution of the project documentation on GitHub. People read the code, contribute to it in the form of pull requests and issues, which in turn prompts the documentation to be changed, at least for the consistently popular projects. So in conclusion, consistent popularity attracts consistent work on documentation. This evolution is different in different project types: library projects need less documentation effort initially than framework projects, but as they gain popularity, frequent changes have to be made to their documentation, reaffirming the conclusions of Dagenais *et al.* [2] on GitHub.

However, many questions still remain unanswered, including whether or not documentation quality improves over time and whether a project can become more competitive based on the consistency and quality of its documentation.

6. REFERENCES

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[2] B. Dagenais and M. P. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 127–136. ACM, 2010.

[3] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pages 233–236, 2013.

[4] M. J. Lee, B. Ferwerda, J. Choi, J. Hahn, J. Y. Moon, and J. Kim. Github developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138. ACM, 2013.

[5] L. Shi, H. Zhong, T. Xie, and M. Li. An empirical study on evolution of api documentation. In *Fundamental Approaches to Software Engineering*, pages 416–431. Springer, 2011.

[6] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 359–364. ACM, 2010.