What's in a GitHub Star?
Understanding Repository Starring Practices in a Social Coding Platform

Hudson Borges

Department of Computer Science, UFMG, Brazil

Marco Tulio Valente

Department of Computer Science, UFMG, Brazil

Abstract

Besides a git-based version control system, GitHub integrates several social coding features. Particularly, GitHub users can *star* a repository, presumably to manifest interest or satisfaction with an open source project. However, the real and practical meaning of *starring a project* was never the subject of an in-depth and well-founded empirical investigation. Therefore, we provide in this paper a throughout study on the meaning, characteristics, and dynamic growth of GitHub stars. First, by surveying 791 developers, we report that three out of four developers consider the number of stars before using or contributing to a GitHub project. Then, we report a quantitative analysis on the characteristics of the top-5,000 most starred GitHub repositories. We propose four patterns to describe stars growth, which are derived after clustering the time series representing the number of stars of the studied repositories; we also reveal the perception of 115 developers about these growth patterns. To conclude, we provide a list of recommendations to open source project managers (e.g., on the importance of social media promotion) and to GitHub users and Software Engineering researchers (e.g., on the risks faced when selecting projects by GitHub stars).

Keywords: GitHub stars, Software Popularity, Social Coding.

1. Introduction

GitHub is the world's largest collection of open source software, with around 28 million users and 79 million repositories.[1] In addition to a `git`-based version control system, GitHub integrates several features for social coding. For example, developers can *fork* their own copy of a repository, work and improve the code locally, and then submit a *pull request* to integrate the changes in the main repository [1–4]. Inspired by the *like* button of modern social networks, GitHub users can also *star* a repository, presumably to manifest interest or satisfaction with the hosted project [5]. However, the real and practical meaning of "starring a project" was never the subject of an in-depth and well-founded empirical investigation.

Furthermore, GitHub's success contributed to the emergence of a competitive open source market. As a result, it is common to see projects competing for the same users. For example, AngularJS, React, and Vue.js compete for developers of JavaScript single-page Web applications. This fact increases the relevance

Email addresses: `hsborges@dcc.ufmg.br` (Hudson Borges), `mtov@dcc.ufmg.br` (Marco Tulio Valente)

[1] `https://github.com/search`, verified on 03/05/2018.

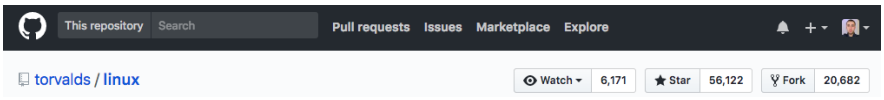


Figure 1: GitHub popularity metrics

of studying the characteristics and practical value of GitHub popularity metrics.

Motivating Survey: In order to provide initial evidence on the most useful metrics for measuring the popularity of GitHub projects, we conducted a survey with Stack Overflow users. We rely on these participants because Stack Overflow is a widely popular programming forum, listing questions and answers about a variety of technologies, which are provided by practitioners with different profiles and background [6]. We randomly selected a sample of 400 Stack Overflow users, using a dump of the site available on Google BigQuery.[2] We e-mailed these users asking then a single question: *How useful are the following metrics to assess the popularity of GitHub projects?* We then presented three common metrics provided by GitHub, which are displayed at the front page of any project: watchers, stars, and forks (see screenshot in Figure 1). Although available on any repository, project owners do not have control over these metrics; any GitHub user can watch, star, or fork a repository, without asking permission to its owners. The survey participants were asked to rank the usefulness of these metrics in a 4-point Likert scale; we also configured the survey system to present the metrics in a random order, to avoid a possible order effect bias. We received 54 answers, which corresponds to a response ratio of 13.5%.

As presented in Figure 2, the results show that stars are viewed by practitioners as the most useful measure of popularity on GitHub, with 83% of answers with scores 3 (31%) or 4 (52%). It is followed by forks with 72% of answers with scores 3-4 (35% and 37%, respectively) and by watchers with 67% (37% and 30%, respectively). Therefore, this initial survey confirms the importance of GitHub stars to practitioners, when compared to forks and watchers. Additionally, stars are often used by researchers to select GitHub projects for empirical studies in software engineering [7–14]. Therefore, a throughout analysis of starring practices can shed light on the properties and risks involved in this selection.

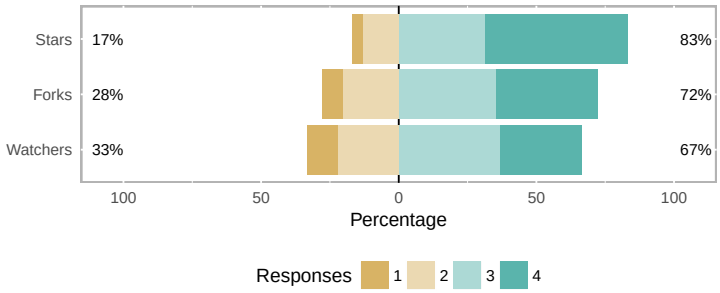


Figure 2: How useful are the following metrics to assess the popularity of GitHub projects? (1: not useful; 4: very useful)

Proposed Study: In a previous conference paper, we started an investigation on the factors and patterns that govern the number of stars of 2,500 GitHub projects [15]. Our first quantitative results indicate that: (i) repositories owned by organizations have more stars than the ones owned by individuals; (ii) there is no

[2] https://cloud.google.com/bigquery/public-data/stackoverflow

correlation between stars and repository's age; but there is a correlation with forks; (iii) repositories tend to receive more stars right after their public release; after this initial period, the growth rate tends to stabilize; (iv) there is an acceleration in the number of stars gained after releases. Furthermore, we showed that the growth of the number of stars is explained by four patterns, which we called *slow*, *moderate*, *fast*, and *viral*.

In this paper, we extend this first study in three major directions:

1. We increment the number of systems from 2,500 to 5,000 public GitHub repositories.
2. We conduct two surveys to shed light on the quantitative results of the initial study. First, we perform a survey with 791 developers to reveal their motivations for starring projects. Ultimately, our intention is to understand *why* developers star GitHub projects. We also conduct a second survey with 115 project owners to reveal their perceptions about the growth patterns proposed in the first study.
3. We investigate the endogenous factors (i.e., the ones that can be extracted directly from a repository, like age) that affect the classification of a project in a given growth pattern. We collect 31 factors along three dimensions and use a machine learning classifier to identify the factors that most distinguish the projects across the proposed growth patterns.

Contributions: Our work leads to at least five contributions:

1. To our knowledge, we are the first to provide solid empirical evidence—both quantitative and qualitative—on the meaning of the number of GitHub stars. Consequently, we recommend that open source maintainers should monitor this metric, as they monitor other project metrics, such as the number of pending issues or pull requests.
2. We reveal that active promotion, particularly on social media sites, has a key importance to increase the number of stars of open source projects. Since these projects are usually maintained by one or two contributors [16], they should allocate time not only to write and maintain the code (developers role) but also to promote the projects (marketing role).
3. We distill a list of threats faced by practitioners and researchers when selecting GitHub projects based on the number of stars. For example, this selection may favor projects with active marketing and advertising strategies, instead of projects following well-established software engineering practices.
4. We implement an open source tool (`http://gittrends.io`) to explore and check our results, including the time series of stars used in this paper and the proposed growth patterns.
5. We provide a public dataset (`https://doi.org/10.5281/zenodo.1183752`) with the application domain of 5,000 GitHub repositories. This dataset can support research in a variety of Software Engineering problems and contexts.

Structure: Section 2 presents and characterizes the dataset used in the study. Section 3 reports GitHub users' major motivations for starring repositories. Section 4 presents a quantitative study on the number of stars of GitHub repositories. Section 5 documents the patterns we propose to describe the growth of the number of stars of GitHub systems. Section 6 investigates factors potentially affecting the inclusion of a repository in the proposed growth patterns. Section 7 describes project owners' perceptions about the growth patterns of their repositories. Threats to validity are discussed in Section 8 and related work is presented in Section 9. We conclude by summarizing our findings and listing future work in Section 10.

2. Dataset

The dataset used in this paper includes the top-5,000 public repositories by number of stars on GitHub. We limit the study to 5,000 repositories for two major reasons. First, to focus on the characteristics of the most starred GitHub projects. Second, because we investigate the impact of application domain on number of stars, which demands a manual classification of each system domain.

All data was obtained using the GitHub API, which provides services to search public repositories and to retrieve specific data about them (e.g., stars, commits, contributors, and forks). The data was collected on January 23rd, 2017. Besides retrieving the number of stars for each system, we also relied on the GitHub API to collect historical data about the number of stars. For this purpose, we used a service from the API that returns all events of a given repository. For each star, these events store the date and the user who starred the repository. However, the GitHub API returns at most 100 events by request (i.e., a page) and at most 400 pages. For this reason, it is not possible to retrieve all stars events of systems with more than 40K stars, as is the case for 18 repositories, such as FreeCodeCamp, Bootstrap, D3, and Font-Awesome. Therefore, these 18 systems are not considered in Sections 5, 6, and 7, since we depend on the complete time series to cluster and derive stars growth patterns.

Table 1 shows descriptive statistics on the number of stars of the repositories in our dataset. The number of stars ranges from 1,596 (for mapnik/mapnik) to 224,136 stars (for FreeCodeCamp/FreeCodeCamp). The median number of stars is 2,866.

Table 1: Descriptive statistics on the number of stars of the repositories in our dataset

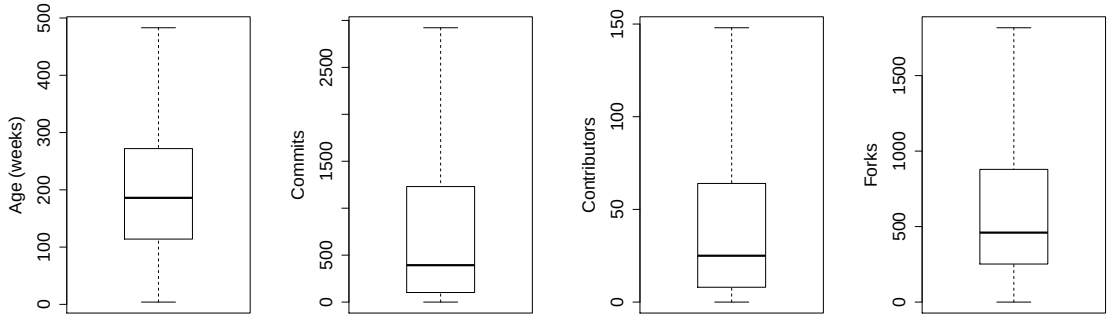
Min	1st Quartile	2nd Quartile	3rd Quartile	Max
1,596	2,085	2,866	4,541	224,136

Age, Commits, Contributors, and Forks: Figure 3 shows boxplots with the distribution of the age (in number of weeks), number of commits, number of contributors, and number of forks for the 5,000 systems in the dataset. For age, the first, second, and third quartiles are 114, 186, and 272 weeks, respectively. For number of commits, the first, second, and third quartiles are 102, 393, and 1,230, respectively. For number of contributors, the first, second, and third quartiles are 8, 25, and 64, respectively;[3] and for number of forks, the first, second, and third quartiles are 252, 460, and 879, respectively. Therefore, the systems in our dataset usually have years of development and many commits and contributors.

Programming Language: As returned by the GitHub API, the language of a project is the one with the highest percentage of source code in its repository. Figure 4 shows the distribution of the systems per programming language. JavaScript is the most popular language (1,559 repositories, 31.1%), followed by Java (520 repositories, 10.4%), Python (441 repositories, 8.8%), Objective-C (374 repositories, 7.4%), and Ruby (305 repositories, 6.1%). Despite a concentration of systems in these languages, the dataset includes systems in 71 languages, including Cuda, Julia, SQLPL, and XSLT (all with just one repository).[4]

[3] We report contributors data as retrieved by the GitHub API. This data may be different from the one presented on the project's page on GitHub, which only counts contributors with GitHub account.

[4] Although HTML is a markup language, it is included in Figure 4. The reason is that we also intend to study repositories containing documentation.



(a) Age (weeks) (b) Commits (c) Contributors (d) Forks

Figure 3: Age, number of commits, number of contributors, and number of forks (outliers are omitted)

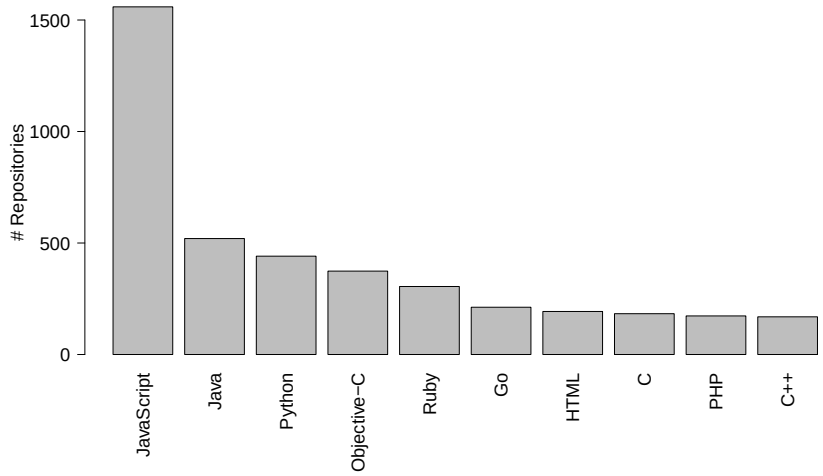


Figure 4: Top-10 languages by number of repositories

Owner: We also characterize our dataset according to repository owner. On GitHub, a repository can be owned by a user (e.g., TORVALDS/LINUX) or by an organization (e.g., FACEBOOK/REACT). In our dataset, 2,569 repositories (51.3%) are owed by users and 2,431 repositories (48.7%) by organizations.

Application Domain: In this study, we also group repositories by application domain. However, different from other source code repositories, like SourceForge, GitHub does not include information about the application domain of a project. For this reason, we manually classified the domain of each system in our dataset. Initially, the first author of this paper inspected the description of the top-200 repositories to provide a first list of application domains, distributed over six domain types, as presented next. These domains were validate with the second paper's author. After this initial classification, the first author inspected the short description, the GitHub page and the project's page of the remaining 4,800 repositories. During this process, he also marked the repositories with dubious classification decisions. These particular cases were discussed by the first and second authors, to reach a consensus decision. To the best of our knowledge, this is the first large-scale classification of application domains on GitHub.

The systems are classified in the following six domains:[5]

1. Application software: systems that provide functionalities to end-users, like browsers and text editors (e.g., WordPress/WordPress and adobe/brackets).
2. System software: systems that provide services and infrastructure to other systems, like operating systems, middleware, and databases (e.g., torvalds/linux and mongodb/mongo).
3. Web libraries and frameworks: systems that are used to implement the front-end (interface) of web-based applications (e.g., twbs/bootstrap and angular/angular.js).
4. Non-web libraries and frameworks: systems that are used to implement other components of an application, despite a web-based interface (e.g., google/guava and facebook/fresco).
5. Software tools: systems that support development tasks, like IDEs, package managers, and compilers (e.g., Homebrew/homebrew and git/git).
6. Documentation: repositories with documentation, tutorials, source code examples, etc. (e.g., iluwatar/java-design-patterns).

Figure 5 shows the number of systems in each domain. The top-3 domains are web libraries and frameworks (1,535 repositories, 30.7%), non-web libraries and frameworks (1,439 repositories, 28.7%), and software tools (972 repositories, 19.4%). The projects in these domains can be seen as meta-projects, i.e., they are used to implement other projects, in the form of libraries, frameworks, or documentation.

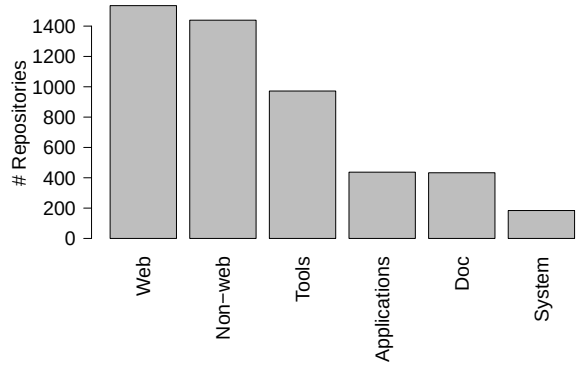


Figure 5: Number of repositories by domain

3. Survey Study

In this section, we describe an investigation with developers to reveal their motivations for starring projects and to check whether they consider the number of stars before using or contributing to projects on GitHub. Section 3.1 describes the design of the survey questionnaire and the selection of the survey participants. Section 3.2 reports the survey results.

[5]This classification only includes first-level domains; therefore, it can be further refined to include subdomains, such Android vs desktop applications.

3.1. Survey Design

The survey questionnaire has two open-ended questions: (1) Why did you star *owner/name*? and (2) Do you consider the number of stars before using or contributing to a GitHub project? In the first question, *owner/name* refers to a repository. Our intention with this question is to understand the motivations behind a developer's decision to star a GitHub repository. With the second question, our goal is to check whether stars is indeed a factor considered by developers when establishing a more close relationship with a project, as a client (or user) or as a contributor (or developer). These questions were sent by email to the last developer who starred each repository in our dataset. The emails were obtained using the GitHub API. When the developers who gave the last star do not have a public email, we select the previous one and so on, successively. We excluded 276 repositories (5.5%) because the last star was given more than six months before the data collection. Therefore, this increases the probability of developers not remembering the concrete reasons they starred these repositories. Moreover, for 336 repositories (6.7%), the selected developer also recently starred other repository in our dataset, thus we excluded these repositories to avoid sending multiple emails to the same developer. Finally, our sample of participants consists of 4,370 developers who recently starred 4,370 repositories from our dataset.

The questionnaire was sent between 13rd and 27th of March 2017. After a period of 30 days, we obtained 791 responses and 173 e-mails returned due to delivery issues (e.g., non-existent recipient), resulting in a response rate of 18.8%. This number of answers represent a confidence interval of 3.15%, for a confidence level of 95%. Considering the locations configured in the respondents' GitHub profile, 133 respondents (16.8%) are from the United States, 74 respondents (9.4%) are from China, 39 (4.9%) are from Brazil, 34 (4.3%) are from Canada, and 27 (3.4%) from India. Other 321 respondents (40.6%) are from 68 different countries and 163 respondents (20.6%) have no location configured in their GitHub profiles. Regarding the respondents' experience in the GitHub platform, their account age ranges from 18 days to 9.12 years, with an average of 4.16 years and a median of 4.09 years. Regarding the programming language used by the participants, 32.6% have most of their public GitHub projects implemented in JavaScript, followed by Python (12.5%), Java (11.8%), Ruby (7.0%), and PHP (5.0%).

To preserve the respondents privacy, we use labels P1 to P791 when quoting the answers. We analyze the answers using thematic analysis [17], a technique for identifying and recording "themes" (i.e., patterns) in textual documents. Thematic analysis involves the following steps: (1) initial reading of the answers, (2) generating a first code for each answer, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes. All steps were performed by the first author of this paper.

3.2. Survey Results

This section presents the answers to the survey questions. A separate subsection discusses each question.

3.2.1. Why did you star owner/name?

In this question, we asked the developers to respond why they starred a given repository. In the next paragraphs, we present four major reasons that emerged after analysing the answers.

To show appreciation: More than half of the participants (52.5%) answered they starred the repositories because they liked the project. In general, the answers mention that stars are used as "likes" button in other social networks, such as Facebook and YouTube. As examples we have:

I liked the solution given by this repo. (P373)

I starred this repository because it looks nice. (P689)

Bookmarking: 51.1% of the participants reported they starred the repositories for later retrieval. We have the following answers as examples:

I starred it because I wanted to try using it later. (P250)

Because I use stars as a "sort of" bookmarks. (P465)

Due to usage: 36.7% of the participants reported they used or are using the project. As examples we have:

I have been using for many years and was about to use again in a new project. (P162)

Because it solved my problem. (P650)

Due to third-party recommendations: 4.6% of the participants starred the repositories due to recommendations from friends, websites, or other developers, as in this answer:

I starred the repository because a technological group recommended it. (P764)

Additionally, five developers (0.6%) answered they do not know or remember the reason why they starred the repositories. Table 2 details the number of answers and the percentage of responses on each theme. Note that one answer can receive more than one theme. For example, the theme *To show appreciation* appeared together with *Bookmarking* and *Due to usage* in 122 and 116 answers, respectively. Moreover, *Due to usage* and *Bookmarking* appeared together in 63 answers.

Table 2: Why do users star GitHub repositories?
(95% confidence level with a 3.15% confidence interval)

Reason	Total	%
To show appreciation	415	52.5
Bookmarking	404	51.1
Due to usage	290	36.7
Due to recommendations	36	4.6
Unknown reasons	5	0.6

Summary: GitHub developers star repositories mainly to show appreciation to the projects (52.5%), to bookmark projects for later retrieval (51.1%), and because they used or are using the projects (36.7%).

3.2.2. Do you consider the number of stars before using or contributing to a project?

In the second question, we asked the participants to respond if they consider the number of stars before using or contributing to GitHub projects.[6] From the 791 answers received in the survey, 14 developers (1.7%) did not answer this specific question. Thus, the numbers presented in this section refer to 777 responses,

[6]Therefore, in this survey, we do not distinguish usage and contribution to Github repositories, which is left for future work.

which gives an updated confidence interval of 3.19%, for a confidence level of 95%. First, we classified the answers in *yes* (the participant does consider the number of stars) and *no* (the participant does not consider the number of stars). As observed in Table 3, 73% of the participants consider the number of stars before using or contributing to GitHub projects and 23.3% answered negatively to this question. Finally, 3.7% of the participants did not correctly answer the question, probably due to a misunderstanding. For example, participant P745 just provided the following answer: "*I am not an active OSS contributor*".

Table 3: Do GitHub users consider the number of stars before using or contributing to a project? (95% confidence level with a 3.19% confidence interval)

Answer	Total	%
Yes	567	73.0
No	181	23.3
Unclear	29	3.7

Positive Answers: Considering the participants who answered positively to this second question, 26.5% commented that the number of stars has a high influence on their decision of using or contributing to a project. As examples, we have these answers:

I always consider the amount of stars on a repo before adopting it in a project. It is one of the most important factors, and in my opinion gives the best metric at a glance for whether a package is production ready. (P365)

Of course stars count is very useful thing, because it tells about project quality. If many people starred something - many people think that it is useful or interesting. (P31)

For 29.3% of the participants who provided a positive answer, the number of stars is just one of the factors they consider before using or contributing to GitHub projects. Other factors include quality of the code/documentation, recent activity, license, and project owner. As examples, we have the following answers:

Yes. I do not take it as my only metric, but having a considerable number of stars and recent activity is reassuring in terms of it being a stable project that my projects can depend on in future. (P104)

I often consider the number of stars (as well as recency of commits, PRs, and issues) in deciding whether to use a project. (P442)

Moreover, 8.8% of the participants consider the number of stars when using but not when contributing to GitHub projects. For example:

I usually contribute more to projects with less stars because of the ease of approach to a smaller community, hence project. On the other hand I normally use frameworks with more stars because of the continuous support they have. (P642)

Additionally, 46 participants (8.1%) provided other comments, as in the following answers:

Yes, a little, I look if it has at least a couple of stars to be sure that doesn't get unmaintained in a short term (P89)

Number of stars is not the major point for me. But it can serve as indicator of something really good (P224)

I don't really notice exactly how many stars something has, but I do notice orders of magnitude (hundreds vs thousands vs tens of thousands) (P421)

Finally, 194 developers (34.2%) did not provide additional information to justify their positive answers.

Negative Answers: Considering only the participants who answered negatively to this second question, 45 participants (24.9%) commented they consider the purpose, domain, and features of the project, but not the number of stars. As examples, we have the answers:

No, my primary interest is: what problem is solving by this project (P203)

Not really. If I like the strategy and implementation, I don't really care how popular or unpopular the repository is (P560)

Moreover, 38 developers (21.0%) answered they consider other measures and sources of information on their decisions, but not the number of stars. For example:

No, I don't consider the number of stars. Number of contributors, commits are important instead of number of stars (P270)

No, I usually know a project from a different source than GitHub itself so I rather refer to the outside opinions on a framework (blogs, articles, community, . . .) on whether it is of good quality than a stars on GitHub (P557)

Additionally, 26 participants (14.3%) provided other reasons for not considering the number of stars (e.g., stars do not reflect project quality); and 74 developers (40.8%) did not provide additional information to justify their answers.

Summary: Three out of four developers consider the number of stars before using or contributing to GitHub projects. Among the developers who consider stars, 29.3% also evaluate other factors, such as source code quality, license, and documentation.

4. Characterization Study

In this section, we describe a quantitative characterization of the number of stars of GitHub projects.[7] More specifically, we provide answers to four research questions:

RQ #1: How the number of stars varies per programming language, application domain, and repository owner? The goal is to provide an initial view about the number of stars of the studied systems, by comparing this measure across programming language, application domain, and repository owner (user or organization).

RQ #2: Does stars correlate with repository's age, number of commits, number of contributors, and number of forks? This investigation can help to unveil possible selection bias that occurs when ranking projects based

[7]This section and the next one are based in our previous conference paper [15], but increasing the number of analysed systems from 2,500 to 5,000 open source projects.

on the number of stars. For example, a positive correlation with repository's age would imply that ranking by stars favors older projects.

RQ #3: How early do repositories get their stars? With this research question, we intend to check whether gains of stars are concentrated in specific phases of a repository's lifetime, specifically in early releases.

RQ #4: What is the impact of new features on stars? This investigation can show if relevant gains in the number of stars in the weeks following new releases.

4.1. Results

RQ #1: How the number of stars varies per programming language, application domain, and repository owner?

Figure 6 shows the distribution of the number of stars for the top-10 languages with more repositories. The top-3 languages whose repositories have the highest median number of stars are: JavaScript (3,163 stars), HTML (3,059 stars), and Go (3,000 stars). The three languages whose repositories have the lowest median number of stars are C (2,679 stars), Java (2,666 stars), and Objective-C (2,558 stars). By applying the Kruskal-Wallis test to compare multiple samples, we found that these distributions differ in at least one language ($p\text{-}value < 0.001$). Then, a non-parametric, pairwise, and multiple comparisons test (Dunn's test) was used to isolate the languages that differ from the others. In Figure 6, the labels a and b in the bars express the results of Dunn's test. Bars sharing the same labels indicate distributions that are not significantly different ($p\text{-}value \leq 0.05$). For example, both JavaScript and HTML share the label b, which means that these distributions have no statistical difference. On the other hand, the distribution with the number of stars of JavaScript projects (label b) is statistically different from Java (label a).

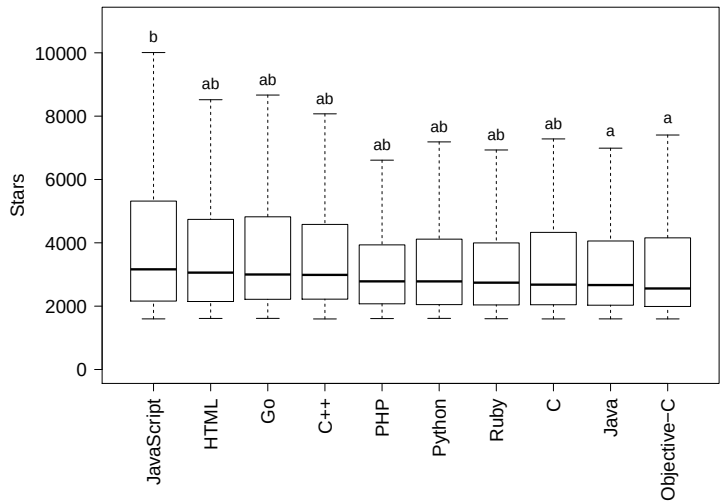


Figure 6: Stars by programming language (considering only the top-10 languages with more repositories)

Figure 7 shows the distribution of the number of stars for the repositories in each application domain. The median number of stars varies as follow: systems software (3,168 stars), applications (3,147 stars), web libraries and frameworks (3,069 stars), documentation (2,942 stars), software tools (2,763 stars), and now-web libraries and frameworks (2,642 stars). By applying the Kruskal-Wallis test, we found that the

distributions are different (*p-value* < 0.001). According to Dunn's test, the distribution of non-web libraries and frameworks (label *c*) is statistically different from all other domains, showing that projects in this domain have less stars. Similarly, tools (label *b*) have more stars only than non-web libraries and frameworks (label *c*). Finally, there is no statistical difference between the number of stars of systems software, applications, web libraries and frameworks, and documentation (since all these distributions have the label *a* in common).

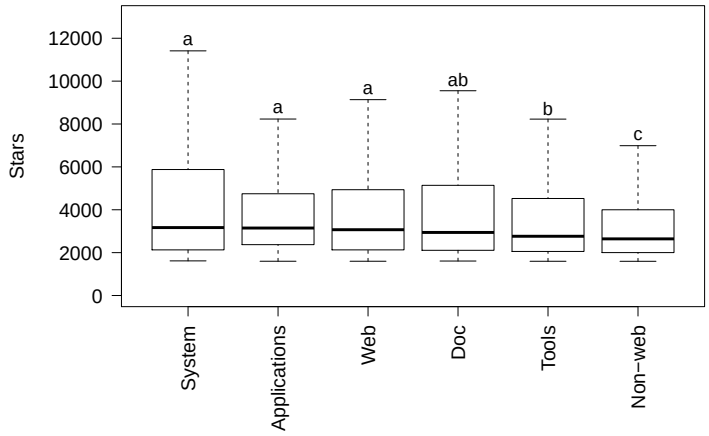


Figure 7: Number of stars by application domain

Finally, Figure 8 shows how the number of stars varies depending on the repository owner (i.e., user or organization). The median number of stars is 3,067 stars for repositories owned by organizations and 2,723 stars for repositories owned by users. By applying the Mann-Whitney test, we detected that these distributions are different (*p-value* < 0.001) with a *very small* effect size (Cohen's $d = -0.178$). Our preliminary hypothesis is that repositories owned by organizations—specifically major software companies and free software foundations—have more funding and resources, which contributes to their higher number of stars.

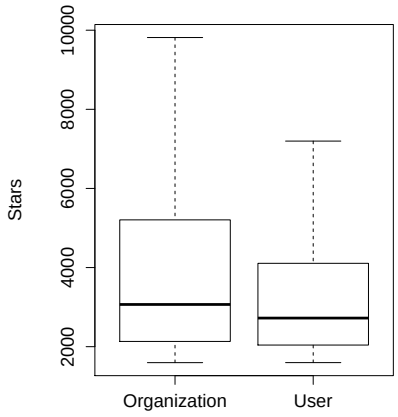
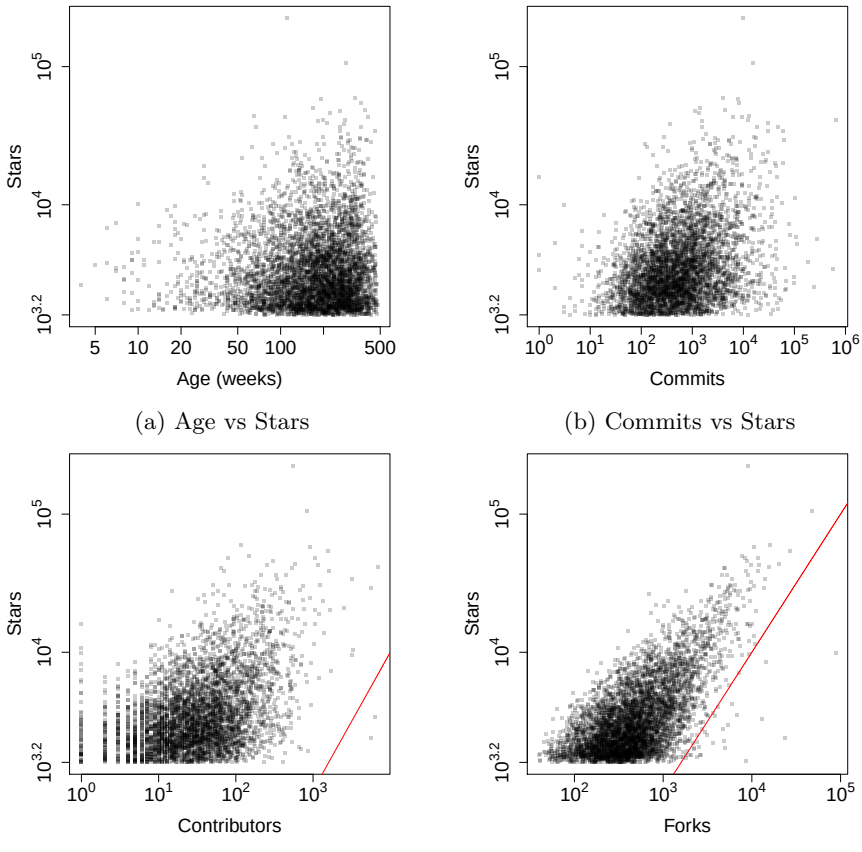


Figure 8: Number of stars by repository owner

Summary: JavaScript repositories have the highest number of stars (median of 3,163 stars); and non-web libraries and frameworks have the lowest number (median of 2,642 stars). Repositories owned by organizations have more stars than the ones owned by individuals.

RQ #2: Does stars correlate with repository's age, number of commits, number of contributors, and number of forks?

Figure 9 shows scatterplots correlating the number of stars with the age (in number of weeks), number of commits, number of contributors, and number of forks of a repository. Following the guidelines of Hinkle et al. [18], we interpret Spearman's rho as follows: $0.00 \leqslant rho < 0.30$ (negligible), $0.30 \leqslant rho < 0.50$ (low), $0.50 \leqslant rho < 0.70$ (moderate), $0.70 \leqslant rho < 0.90$ (high), and $0.90 \leqslant rho < 1.00$ (very high). First, the plots suggest that stars are not correlated with the repository's age (Figure 9a). We have old repositories with few stars and new repositories with many stars. For example, FACEBOOKINCUBATOR/CREATE-REACT-APP has only five months and 19,083 stars, while MOJOMBO/GRIT has more than 9 years and 1,883 stars. Essentially, this result shows that repositories gain stars at different speeds. We ran Spearman's rank correlation test and the resulting correlation coefficient is close to zero ($rho = 0.050$ and $p\text{-}value < 0.001$).



(a) Age vs Stars

(b) Commits vs Stars

(c) Contributors vs Stars

(d) Forks vs Stars

Figure 9: Correlation analysis. In subfigures (c) and (d), the line is the identity relation

The scatterplot in Figure 9b shows that stars have a low correlation with number of commits ($rho = 0.439$ with $p\text{-}value < 0.001$). However, as presented in Figure 9c, stars have a moderate correlation with contributors ($rho = 0.502$ with $p\text{-}value < 0.001$). In this figure, a logarithm scale is used in both axes; the line represents the identity relation: below the line are the systems with more contributors than stars. Interestingly, two systems indeed have more contributors than stars: raspberrypi/linux (6,277 contributors and 3,414 stars) and Linuxbrew/legacy-linuxbrew (5,681 contributors and 2,397 stars). This happens because they are forks of highly successful repositories (torvalds/linux and Homebrew/brew, respectively). The top-3 systems with more stars per contributor are shadowsocks/shadowsocks (16,017 stars/contributor), wg/wrk (10,658 stars/contributor), and octocat/Spoon-Knife (9,961 stars/contributor). However, these systems have just one contributor. The three systems with less stars per contributor are DefinitelyTyped/DefinitelyTyped (2.97 stars/contributor), nodejs/node-convergence-archive (2.88 stars/contributor), and openstack/nova (2.23 stars/contributor).

Finally, Figure 9d shows plots correlating stars and forks. As suggested by the followed guidelines, there is a moderate positive correlation between the two measures ($rho = 0.558$ and $p\text{-}value < 0.001$). For example, twbs/bootstrap is the second repository with the highest number of stars and also the second one with more forks. angular/angular.js is the fifth repository in number of stars and the third one with more forks. In Figure 9d, we can also observe that only 28 systems (0.56%) have more forks than stars. As examples, we have a repository that just provides a tutorial for forking a repository (octocat/SpoonKnife) and a popular puzzle game (gabrielecirulli/2048), whose success motivated many forks with variations of the original implementation.

Summary: There is no correlation between stars and repository's age; however, there is a low correlation with commits, and a moderate correlation with contributors and forks.

RQ #3: How early do repositories get their stars?

Figure 10 shows the cumulative distribution of the time fraction a repository takes to receive at least 10%, at least 50%, and at least 90% of its stars. Around 32% of the repositories receive 10% of their stars very early, in the first days after the initial release (label A, in Figure 10). We hypothesize that many of these initial stars come from early adopters, who start commenting and using novel open source software immediately after they are public released.[8] After this initial burst, the growth of the number of stars tend to stabilize. For example, half of the repositories take 48% of their age to receive 50% of their stars (label B); and around half of the repositories take 87% of their age to receive 90% of their number of stars (label C).

Figure 11 shows the distribution of the fraction of stars gained in the first and last four weeks of the repositories. For the first four weeks, the fraction of stars gained is 0.4% (first quartile), 7.0% (second quartile), and 21.6% (third quartile). For the last four weeks, it is 0.8% (first quartile), 1.6% (second

[8]It is worth mentioning that GitHub repositories can be created private and turned public later. In this RQ, we consider the latter event, which we referred as public release.

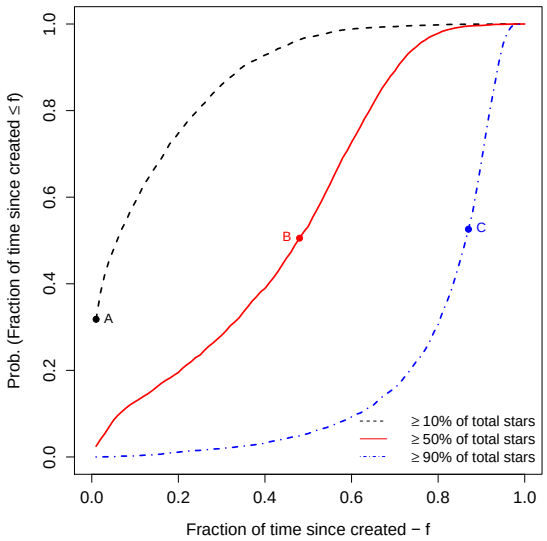


Figure 10: Cumulative distribution of the time fraction a repository takes to receive 10%, 50%, and 90% of its stars

quartile), and 2.7% (third quartile). By applying the Mann-Whitney test, we found that these distributions are different ($p\text{-}value < 0.001$) with a *large* effect size (Cohen's $d = 0.856$).

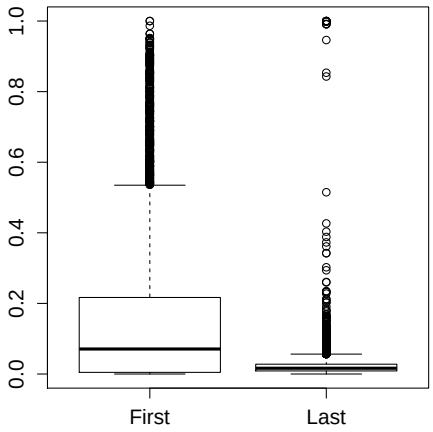


Figure 11: Fraction of stars gained in the first four weeks and in the last four weeks

Summary: Repositories have a tendency to receive more stars right after their public release. After that, the growth rate tends to stabilize.

RQ #4: What is the impact of new features on stars?

In this research question, we investigate the impact of new features on the number of stars of GitHub repositories. The goal is to check whether the implementation of new features (resulting in new releases of the projects) contribute to a boost in the number of stars. Specifically, we selected 1,539 repositories from our

dataset (30.7%) that follow a semantic versioning convention to number releases. In such systems, versions are identified by three integers, in the format $x.y.z$, with the following semantics: increments in x denote major releases, which can be incompatible with older versions; increments in y denote minor releases, which add functionality in a backward-compatible manner; and increments in z denote bug fixes. In our sample, we identified 1,304 major releases and 8,570 minor releases.

First, as illustrated in Figure 12, we counted the fraction of stars received by each repository in the week following all releases (FS_{All}) and just after major releases (FS_{Major}). As mentioned, the goal is to check the impact of new features in the number of stars right after new releases (however, in the end of the RQ, we also consider the impact of different week intervals). As an example, Figure 13 shows the time series for REPORTR/DASHBOARD, using dots to indicate the project's releases (v1.0.0/v.1.1.0, v2.0.0, and v2.1.0, respectively). This project has $FS_{\text{All}} = 0.525$ (i.e., 52.5% of its stars were gained in the weeks following the four releases) and $FS_{\text{Major}} = 0.248$ (i.e., 24.8% of its stars were gained in the weeks following the releases v1.0.0 and v2.0.0).

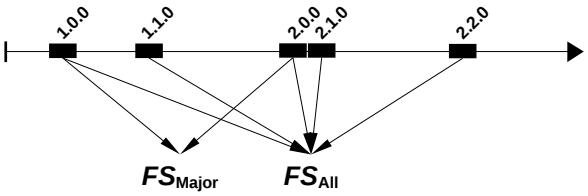


Figure 12: Fraction of stars for all releases (FS_{All}) and just after major releases (FS_{Major})

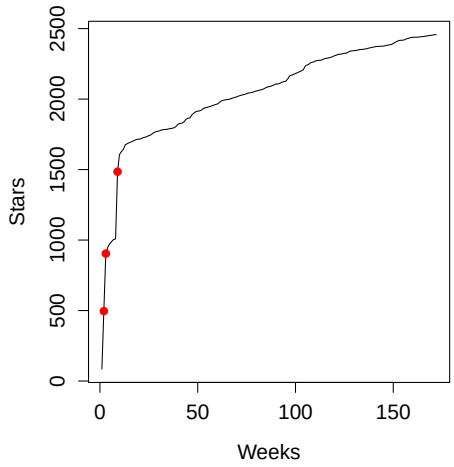


Figure 13: REPORTR/DASHBOARD (the dots indicate weeks with releases)

Figure 14 shows the distribution of FS_{All} and FS_{Major} for all selected repositories. When considering all releases, the fraction of stars gained in the first week after the releases is 1.0% (first quartile), 3.1% (second quartile), and 10.5% (third quartile). For the major releases, it is 0.5% (first quartile), 1.2% (second quartile), and 3.8% (third quartile). By applying the Mann-Whitney test, we found that these distributions are different ($p\text{-}value < 0.001$), but with a *small* effect size (Cohen's $d = 0.316$). YARNPKG/YARN (a package manager for JavaScript) is the repository with the highest fraction of stars received after releases. The repository has one year, 21,809 stars, and gained most of its stars (83.0%) in the weeks after its releases.

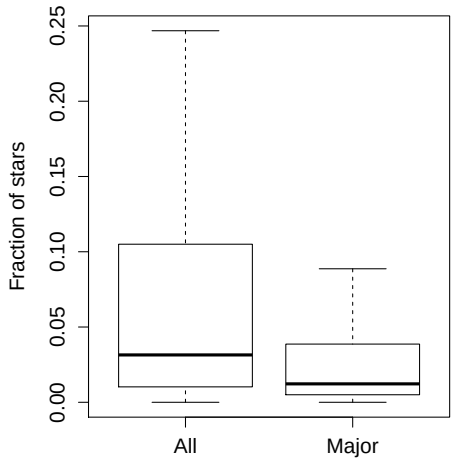


Figure 14: Fraction of stars gained in the first week after all releases and just after the major releases

We also computed two ratios: $R_{\text{All}} = FS_{\text{All}}/FT_{\text{All}}$ and $R_{\text{Major}} = FS_{\text{Major}}/FT_{\text{Major}}$, where FT is the fraction of time represented by the weeks following the releases per the repository's age. When $R_{\text{All}} > 1$ or $R_{\text{Major}} > 1$, the repository gains proportionally more stars after releases. For example, REPORTR/DASHBOARD (Figure 13) has $FT_{\text{All}} = 0.019$ (i.e., the weeks following all releases represent only 1.9% of its total age) resulting in $R_{\text{All}} = 0.525/0.019 = 27.047$. Therefore, releases have a major impact on its number of stars. Figure 15 shows boxplots with the results of R_{All} and R_{Major} for all repositories. Considering all releases, we have that R_{All} is 0.89 (first quartile), 1.35 (second quartile), and 2.20 (third quartile). For major releases only, we have that R_{Major} is 0.83 (first quartile), 1.49 (second quartile), and 3.37 (third quartile). By applying the Mann-Whitney test, we found that these distributions are different (*p-value* < 0.05); but after computing Cohen's d, we found a *very small* effect size ($d = -0.188$).

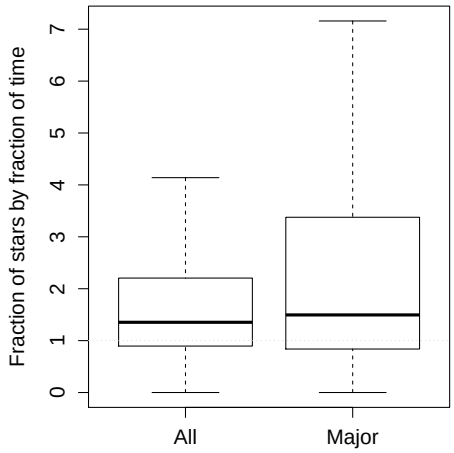


Figure 15: Fraction of stars gained in the week following all releases (or just the major releases) / fraction of time represented by these weeks

Finally, Figure 16 shows the median values of R_{All} and R_{Major} computed using stars gained after n weeks ($1 \leq n \leq 4$). Both ratios decrease (for major and all releases). Therefore, although there is some gains of stars after releases, they tend to decrease after few weeks.

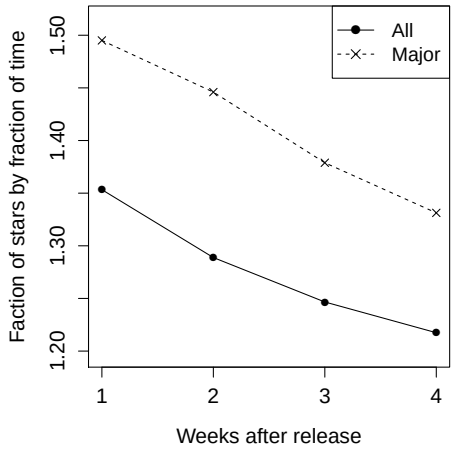


Figure 16: Fraction of stars by fraction of time (median values), computed using different time intervals

Summary: There is an acceleration in the number of stars gained after releases. For example, half of the repositories gain at least 49% more stars in the week following major releases than in other weeks. However, because repositories usually have more weeks without releases, this phenomenon is not sufficient to result in a major concentration of stars after releases. For example, 75% of the systems gain at most 3.8% of their stars in the week following major releases.

Implications for Empirical Software Engineering Researchers: Regarding the selection of GitHub projects for empirical studies based on number of stars, the following observations are derived from our findings: (1) this selection favors JavaScript systems (31.1% of the systems in our dataset) and also web libraries and frameworks (30.7% of the dataset systems); (2) this selection might result in a relevant number of projects that are not software systems (8.6% of the projects in our dataset are tutorials, books, *awesome-lists*, etc); (3) this selection favors large projects (in terms of number of contributors) with many forks, as we concluded when investigating RQ #2 (correlation analysis); (4) additionally, after examining RQ #3, we recommend researchers (and also practitioners) to check whether the stars are not gained in a short time interval, for example, after the project public release.

5. Stars Growth Patterns

In this section, we investigate common growth patterns concerning the number of stars of the GitHub repositories in our dataset. To this purpose, we use the KSC algorithm [19]. This algorithm uses an iterative approach, similar to the classical K-means clustering algorithm, to assign the time series in clusters and then refine the clusters centroids by optimizing a specific time series distance metric that is invariant to scaling and shifting. As result, the clusters produced by the KSC algorithm are less influenced by outliers. KSC is used in other studies to cluster time series representing the popularity of YouTube videos [20] and Twitter posts [21]. Like K-means [22], KSC requires as input the number of clusters k.

Because the time series provided as input to KSC must have the same length, we only consider data regarding the last 52 weeks (one year). We acknowledge that this decision implies a comparison of projects in different stages of their evolution (e.g., a very young project, which just completed one year, and mature

projects, with several years). However, it guarantees the derivation of growth patterns explaining the dynamics of the most recent stars received by a project and in this way it also increases the chances of receiving valuable feedback of the projects contributors, in the survey described in Section 7. Due to this decision, we had to exclude from our analysis 333 repositories (6.6%) that have less than 52 weeks.

We use the β_{CV} heuristic [23] to define the best number k of clusters. β_{CV} is defined as the ratio of two coefficients: variation of the intracluster distances and variation of the intercluster distances. The smallest value of k after which the β_{CV} ratio remains roughly stable should be selected. This means that new added clusters affect only marginally the intra and intercluster variations [24]. In our dataset, the values of β_{CV} start to stabilize for $k = 4$ (see Figure 17). Note that although the value of β_{CV} increases for $k = 5$ (from 0.968 to 1.002, respectively), the β_{CV} for $k = 4$ remains almost the same for $k = 6$ and $k = 7$ (0.966 and 0.963, respectively). For this reason, we use four clusters in this study.

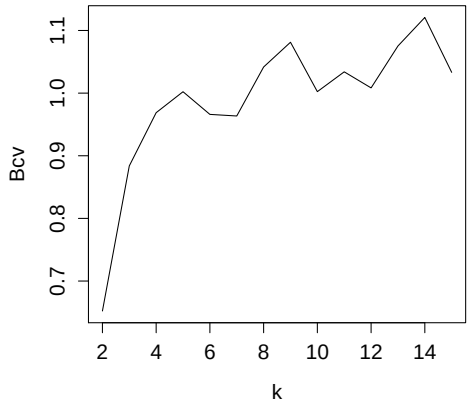
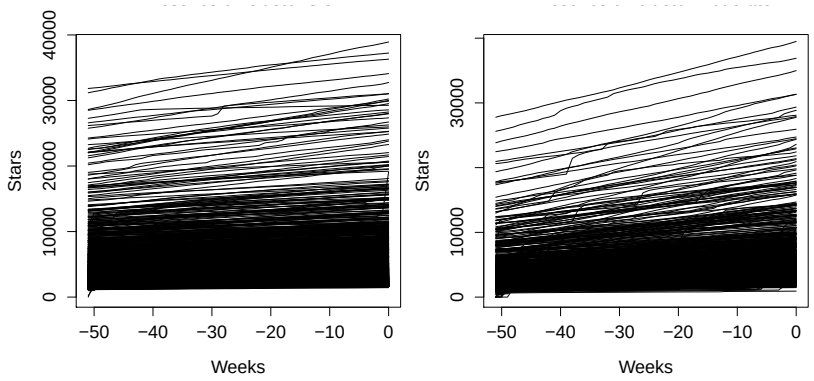


Figure 17: β_{CV} for $2 \leq k \leq 15$

5.1. Proposed Growth Patterns

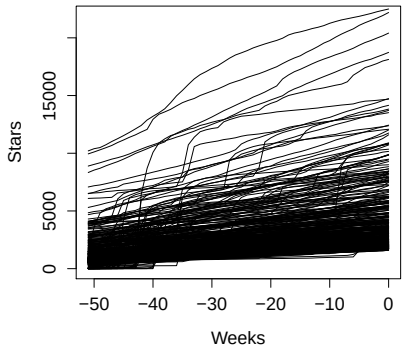
Figure 18 shows plots with the time series in each cluster. The time series representing the clusters' centroids are presented in Figure 19. The time series in clusters C1, C2, and C3 suggest a linear growth, but at different speeds. On the other hand, the series in cluster C4 suggest repositories with a sudden growth in the number of stars. We refer to these clusters as including systems with *Slow*, *Moderate*, *Fast*, and *Viral* Growth, respectively.

Slow growth is the dominant pattern, including 58.2% of the repositories in our sample, as presented in Table 4. The table also shows the percentage of stars gained by the cluster's centroids in the period under analysis (52 weeks). The speed in which the repositories gain stars in cluster C1 is the lowest one (19.8% of new stars in one year). Moderate growth is the second pattern with more repositories (30.0% of the repositories and 63.9% of new stars in one year). 9.3% of the repositories have a fast growth (218.6% of new stars in the analyzed year). Cluster C4 (Viral Growth) includes repositories with a massive growth in their number of stars (1,317%). However, it is a less common pattern, including 2.3% of the repositories. Figure 20 shows two examples of systems with a viral growth: CHRISLGARRY/APOLLO–11 (Apollo 11 guidance computer source code, with a peak of 19,270 stars in two weeks) and NAPTHA/TESSERACT.JS (a JavaScript library to recognize words in images, which received 6,888 stars in a single week).

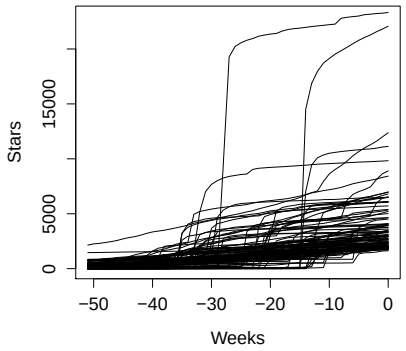


(a) Cluster C1

(b) Cluster C2



(c) Cluster C3



(d) Cluster C4

Figure 18: Clusters of time series produced by the KSC algorithm

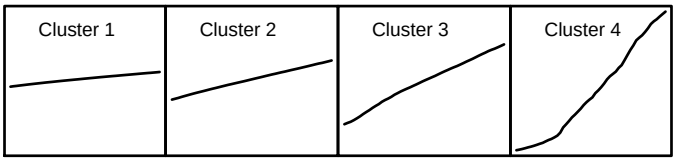
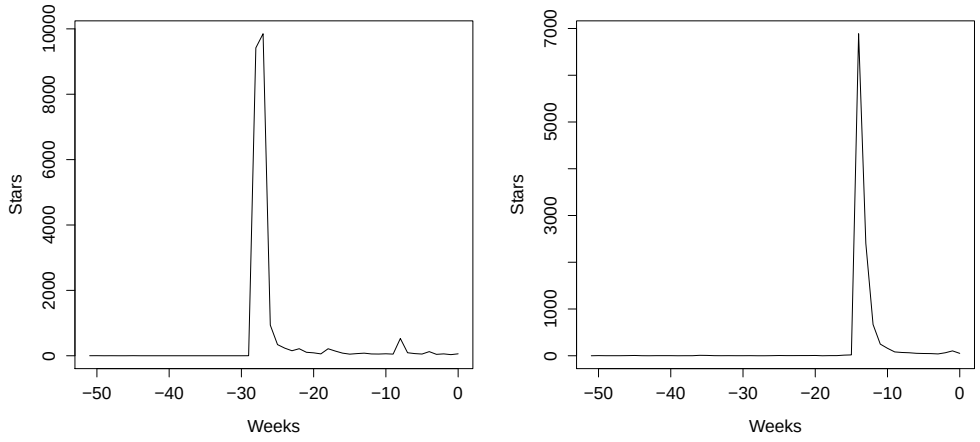


Figure 19: Time series representing the centroids of each cluster

Table 4: Stars Growth Patterns

Cluster	Pattern	# Repositories	Growth (%)
C1	Slow	2,706 (58.2%)	19.8
C2	Moderate	1,399 (30.0%)	63.9
C3	Fast	434 (9.3%)	218.6
C4	Viral	110 (2.3%)	1,317.2

We also investigate the correlation of the proposed growth patterns with the repositories ranking by number of stars. To this purpose, we calculate the ranking of the studied repositories on week 0 (first week) and 51 (last week), by number of stars. Next, we calculate the repositories rank in such weeks. Repositories with positive values improved their ranking position, whereas negative values mean repositories losing positions. Figure 21 presents the distribution of the rank differences by growth pattern. Initially, we can observe that at least 75% of the slow repositories dropped in the ranking. By contrast, almost all



(a) Apollo-11 (b) Tesseract.js

Figure 20: Examples of systems with viral growth

repositories (109 out of 110) with viral growth improved their rank in the same period. Finally, 82% and 96% of the repositories with moderate and fast growth, respectively, increased their ranks. By applying a Kruskal-Wallis test, we found that these distributions are different ($p\text{-}value < 0.001$). According to Dunn's test, the rank differences of repositories with slow and moderate growth are statistically different from the other patterns; however, there is no statistical difference between repositories with fast and viral growth.

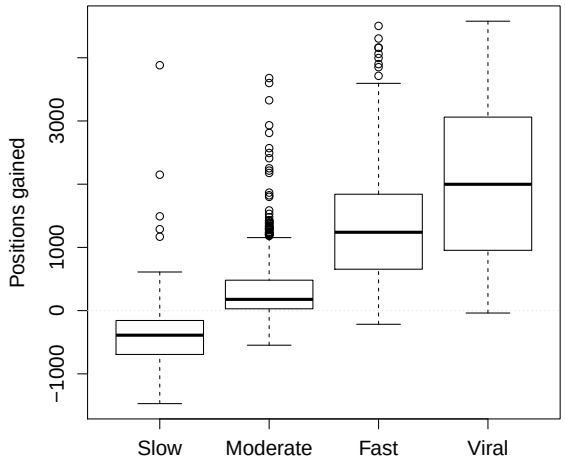


Figure 21: Rank differences in the interval of one year

6. Growth Patterns Characterization

In this section, we identify endogenous factors that distinguish the repositories in each growth pattern. Revealing these factors is important because developers can strive to improve or change the ones that can be controlled or better understand the impact of those they have no control.

6.1. Methodology

To identify the most influential factors, we collected a set of characteristics of the repositories following each proposed growth pattern and applied a Random Forest classifier [25]. We selected Random Forest because it is robust to noise and outliers [26–28].

Table 5 lists 31 factors along three dimensions potentially affecting the stars growth of the repositories. The Repository dimension includes factors that are accessible to users on the repositories' page in GitHub. Usually, these pages are the main, or even unique, source of information about the projects and might influence the developers' decision on using (or not) a project. For example, forks, and subscribers are measures of potential contributors to the repository. Moreover, the quality of README files is another criterion considered by developers when selecting projects [5].

Table 5: Factors potentially affecting the growth pattern of a repository

Dimension	Factor	Description
Repository	Stars (r.stars)	Number of stars
	Forks (r.forks)	Number of forks
	Network (r.network)	Number of repositories in the network[9]
	Subscribers (r.subscribers)	Number of users registered to receive notifications
	Age (r.age)	Number of weeks since creation
	Last Push (r.pushed)	Number of weeks since last *git push*
	Is Fork (r.is_fork)	Repository is a fork (boolean value)
	Has homepage (r.has_homepage)	Repository has a homepage (boolean value)
	Size (r.size)	Size of the repository in MB
	Language (r.language)	Main programming language of the repository
	Has Wiki (r.has_wiki)	Repository has Wiki (boolean value)
	Has Pages (r.has_pages)	Repository has GitHub pages[10] (boolean value)
	Is Mirror (r.mirror)	Repository is a mirror (boolean value)
	Domain (r.domain)	Application domain (as defined in Section 2)
	Description length (r.description_length)	Number of words in the description
	README length (r.readme_length)	Number of words in the README file
Owner	Account Type (o.type)	Account type: User or Organization[11]
	Company (o.company)	Owner belongs to an organization (boolean value)
	Has Public Email (o.email)	Owner has a public email (boolean value)
	Public Repositories (o.repos)	Number of public repositories
	Public Gists (o.gists)	Number of public code snippets
	Followers (o.followers)	Number of followers
	Following (o.followings)	Number of following
	Total stars (o.stars)	Sum of all stars of all public repositories
	Account Age (o.age)	Number of weeks since its account was created
Activity (last 52 weeks)	Commits (a.commits)	Number of commits
	Contributors (a.contributors)	Number of contributors
	Tags (a.tags)	Number of git tags
	Releases (a.releases)	Number of releases
	Issues (a.issues)	Number of issues
	Pull Requests (a.pull_requests)	Number of pull requests

[9] Total number of forks including forks of forks.

[10] `https://pages.github.com`

[11] `https://help.github.com/articles/what-s-the-difference-between-user-and-organization-accounts`

The OWNER dimension includes factors related to the repository' owner, for example, number of followers and account type. For example, developers with more followers may take advantage of GitHub News Feed[12], since their recent activities are shown to more developers [29]. Finally, developers owning popular repositories (by number of stars) might also attract more users to their other projects.

The ACTIVITY dimension includes factors related to the coding activity in the 52 weeks considered when extracting the growth patterns. For example, higher number of commits might indicate that the project is in constant evolution whereas number of contributors, issues, and pull requests might indicate the engagement of the community with the project.

Before using the Random Forest classifier, we performed a hierarchical cluster analysis on the 31 features in Table 5. This technique is proposed for assessing features collinearity and it is used in several other studies [27, 30]. Figure 22 presents the final hierarchical cluster. For sub-hierarchies with correlation greater than 0.7, only one variable was selected to the classifier. For this reason, we removed the features *a.pull_requests* and *a.contributors* (first cluster below the line), *r.network* (second cluster), and *o.type* (third cluster).

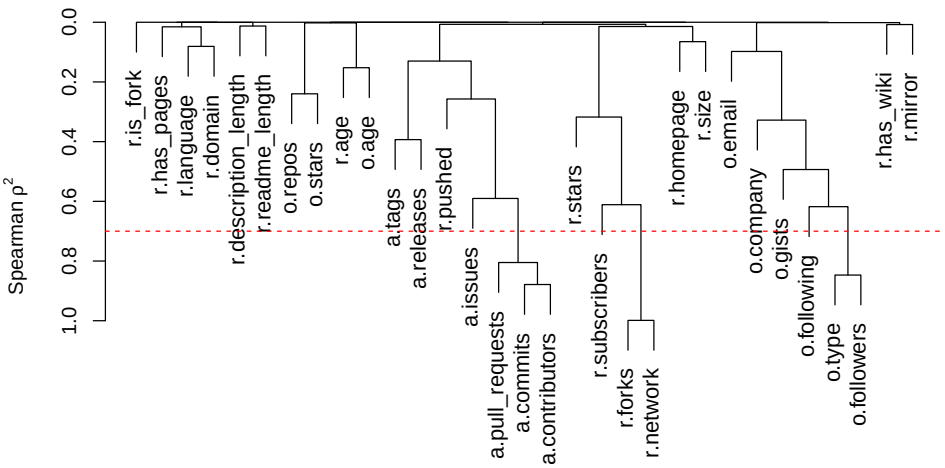


Figure 22: Correlation analysis (as result, we removed features *a.pull_requests*, *a.contributors*, *r.network*, and *o.type*)

6.2. Most Influential Factors

To assess the relative importance of the selected features in discriminating each growth pattern, we used the `rfPermute` package for `R` [31]. We use the Mean Decrease Accuracy (MDA), which is determined during the prediction error measure phase, to rank the features based on their importance to the classifier. MDA is quantified by measuring the change in prediction accuracy, when the values of the features are randomly permuted compared to the original observations [32].

Table 6 lists the top-10 most influential factors according to the feature importance ranking (all of them with *p-value* < 0.01). As we can observe, these features are spread among the three dimensions, which shows their importance. For REPOSITORY, the two most discriminative features are *Age* and *Last Push*, respectively. In fact, for *Age*, we observed that *slow* growth is more common in old repositories whereas repositories presenting *fast* and *viral* growth are newest. The median number of weeks since creation is 235

[12] `https://help.github.com/articles/news-feed`, a dashboard with recent activity on repositories.

for *slow*, 167 for *moderate*, 96 for *fast*, and 76 for *viral*. Regarding *Last Push*, we observed long inactive periods in repositories with *slow* growth. The median number of weeks since the last code update is 3.53 for *slow*, 0.80 for *moderate*, 0.52 for *fast*, and 0.49 for *viral*. For the OWNER dimension, the two most discriminative features are *Account Age* and *Followers*, respectively. The owners of repositories with *viral* growth have the lowest account age (median of 173 weeks) and the lowest median number of followers (0). Finally, for ACTIVITY, the two most discriminative features are *Issues* and *Commits*, respectively. Similarly to previous factors, repositories with *slow* growth have the lowest number of commits (only 19 commits). Moreover, *moderate* and *fast* repositories have higher median number of issues than *slow* and *viral* repositories (51, 64, 19, and 11 issues, respectively).

Table 6: Top-10 most influential factors ($p\text{-}value < 0.01$)

Ranking	Factor	Dimension	Actionable
1	Age (r.age)	Repository	-
2	Last Push (r.pushed)	Repository	Yes
3	Issues (a.issues)	Activity	-
4	Commits (a.commits)	Activity	Yes
5	Forks (r.forks)	Repository	-
6	Account Age (o.age)	Owner	-
7	Stars (r.stars)	Repository	-
8	Subscribers (r.subscribers)	Repository	-
9	Followers (o.followers)	Owner	-
10	Tags (a.tags)	Repository	Yes

Although some factors cannot be controlled by developers, others depend on their actions. From the top-10 most influential factors in Table 6, three are directly impacted by developers' actions (column "Actionable"). These results suggest that projects with frequent updates (*Last Push*), a rich development history (*Commits*), and frequent releases (*Tags*) tend to attract more attention, in terms of number of stars. However, it is also important to highlight that "correlation does not necessarily imply in causation". Therefore, it might be the project popularity that triggers constant pushes, commits, and releases. In other words, these results indicate that success in open source projects has its own price, which comes in the form of constantly having to update and improve the projects. Developers should be aware of this fact and reserve time to maintain a successful project. In fact, a recent survey shows that lack of time is the third most common reason for the failure of modern open source projects [33].

Finally, to assess the effectiveness of the classifier, we relied on metrics commonly used in Machine Learning and Information Retrieval [34]. Precision measures the correctness of the classifier in predicting the repository growth pattern. Recall measures the completeness of the classifier in predicting growth patterns. F-measure is the harmonic mean of precision and recall. Table 7 shows the results for each growth pattern and the overall result. In general, Random Forest performed satisfactorily for all patterns with a precision of 65.81%, recall of 68.40%, and F-measure of 67.08%. The *Slow* pattern, which concentrates most of the repositories, presented the most accurate results (F-measure = 81.47%). On the other hand, Viral has the worst results (F-measure = 6.61%), which can be caused by exogenous factors that are hard to predict.

Table 7: Classification effectiveness

Growth Pattern	Precision (%)	Recall (%)	F-measure (%)
Slow	75.98	87.80	81.47
Moderate	54.16	47.96	50.87
Fast	47.43	29.72	36.54
Viral	36.36	3.64	6.61
Overall	65.81	68.40	67.08

Summary: When we compare the proposed growth patterns, *Age* is the most discriminative feature, followed by number of *Issues* and *Last Push*. Moreover, three out of four features from the ACTIVITY dimension are in the top-10 most discriminative ones, which confirms the importance of constantly maintaining and evolving open source projects.

7. Developers' Perceptions on Growth Patterns

In this section, we describe a survey with developers to reveal their perceptions on the growth patterns proposed in this work. Section 7.1 describes the design of the survey questionnaire and the selection of the survey participants. Section 7.2 reports the survey results.

7.1. Survey Design

In this second survey, we asked developers to explain the reasons for the *slow*, *moderate*, *fast*, or *viral* growth observed in the number of stars of their repositories. The questionnaire was sent by email to the repository's owner, for repositories owned by *Users*, or to the contributor with the highest number of commits, for repositories owned by *Organizations*. For each growth pattern, we randomly selected 100 repositories whose developers have a public email. Exceptionally for repositories classified with *viral* growth, we selected 45 developers because they are the only ones with public emails on GitHub. Thus, our sample of participants consists of 345 developers.

The questionnaire was sent between the 18th to 22nd of May 2017. After a period of seven days, we received 115 responses, resulting in a response ratio of 33.3%, considering the four growth patterns together (see details in Table 8). To preserve the respondents privacy, we use labels R1 to R115 when quoting their answers. After receiving the answers, the first paper's author analyzed them, following the same steps of the survey presented in Section 3.

Table 8: Number of survey participants and answers per growth pattern
CI = Confidence interval at confidence level of 95%

Growth Pattern	Participants	Answers	%	CI
Slow	100	26	26.0	19.1
Moderate	100	33	33.0	16.9
Fast	100	34	34.0	16.1
Viral	45	22	48.9	18.8

7.2. Survey Results

Table 9 lists five major reasons for *slow* growth, according to the surveyed developers. Unmaintained or low activity was the main reason, reported by 14 developers (53.8%). Limited or lack of promotion was mentioned by four developers (15.3%). For three developers, the project focus on a specific niche audience, thus not being so popular as other repositories. Furthermore, emergence of alternative solutions was the reason pointed by two developers. Other three developers reported they have no idea on the reasons of the *slow* growth. Finally, five developers provided other reasons (e.g., project age). Examples of answers include:

The reason is there's no new material there. Also the material that is there is becoming outdated and less relevant over time. (R27, Unmaintained or low activity)

I believe the primary reason is that I am doing virtually nothing to actively promote the project. (R26, Limited or lack of promotion)

I don't know the root cause, my guess is that it's a rather specialized tool with a limited audience. (R38, Niche audience)

Table 9: Reasons for Slow Growth
(95% confidence level with a 19.1% confidence interval)

Reason	Answers	Percentage (%)
Unmaintained or low activity	14	53.8
Limited or lack of promotion	4	15.3
Niche audience	3	11.5
Alternative solutions	2	7.6
Unknown	3	11.5
Other reasons	5	19.2

After analyzing the reasons for *moderate* growth, we identified two conflicting sentiments in the answers: (a) *positives* reasons, which are contributing to the stars growth; (b) *negative* reasons, which are limiting the stars growth. Table 10 lists the major reasons for the *positive* and *negative* sentiments.

For *positive* sentiments, 15 developers (45.4%) mentioned active promotion (mainly on social media sites, as Hacker News[13]). The use of trending technologies was mentioned by nine developers (27.2%). For example, DANIALFARID/NG-FILE-UPLOAD (a popular ANGULAR component) is benefited by the large community of ANGULAR practitioners. Active project (e.g., with frequent updates and fast issues resolution) was mentioned by seven developers (21.2%). Three developers explicitly mentioned the repository provides an innovative solution and two developers mentioned that code or documentation quality contributed to the stars growth. Finally, three other positive reasons were provided (project usability, usefulness, and maturity). As examples we have these positive answers:

It could be related to how many people are using Angular JS and the development and new features in the module had been active for couple years. (R34, Trending technology, Active project)

The initial increase in stars happened as word of the project got out. I initially had a Product Hunt page and posted it on Hacker News. From there it is started to popup on other tech sites. (R85, Active promotion)

[13] https://news.ycombinator.com

Our continued releases every 3-4 months for nearly 6 years is probably the reasoning. We are a steady, stable, open source solution for reverse engineering. (R16, Active project, Maturity)

Table 10: Reasons for Moderate Growth
(95% confidence level with a 16.9% confidence interval)

Positive Sentiments			Negative Sentiments		
Reason	Answers	Percentage (%)	Reason	Answers	Percentage (%)
Active promotion	15	45.4	Niche audience	3	9.0
Trending technology	9	27.2	Low activity	2	6.0
Active project	7	21.2	Limited or lack of promotion	1	3.0
Innovative project	3	9.0	Old project	1	3.0
Code or doc. quality	2	6.0			
Other	3	9.0			

For answers transmitting *negative* sentiments, three developers mentioned the project's niche audience as a restrictive growth factor. Moreover, low activity and limited or lack of promotion were mentioned by two and one developers, respectively. Finally, one developer mentioned that the project age is restricting its stars growth. Examples of negative answers are:

I think the demographics for [repository] users shifts towards the [other-repository] – new devs and people new to a young language tend to look for more features, and [repository] is explicitly not that. (R25, Niche audience)

My best guess is that it's an older project that's occasionally attracting new people, but there's no single big "marketing event" where it gets a huge spike of GitHub stars. (R28, Old project, Limited or lack of promotion)

For repositories presenting *fast* growth, Table 11 lists six major reasons reported by their developers. Active promotion is the major reason according to 22 developers (64.7%). Furthermore, trending technology was mentioned by 11 developers (32.3%). Other eight developers (24.5%) mentioned that it is an innovative project. Examples of reasons for *fast* growth include:

It's a popular project because nothing else like it exists for React. (R72, Innovative project, Trending technology)

We've been adding a lot of features in the last year, and I've been trying to evangelise the project to gain new users - some of those things probably helped a lot. (R66, Active project, Active promotion)

Finally, Table 12 lists five major reasons that emerged after analysing the developers' answers for *viral* growth. As observed, 16 developers (72.7%) linked this behavior to successful posts in social media sites, mostly Hacker News. Code or documentation quality were mentioned by six developers (27.2%). Four developers (19.0%) linked the *viral* growth to trending technologies. As examples of answers we have:

Yes, we had a huge bump in stars. The secret: coverage by Hacker News, which resulted in follow-up by other news sites. (R44, Promotion on social media sites)

In my opinion is just that [repository] replied to some people need and gain adoption very fast. Sharing the project on reddit/twitter/hacker news helped a lot the spread of it. In my opinion the quality of docs/examples helps a lot. (R103, Promotion on social media sites, Code or documentation quality, Useful project)

Table 11: Reasons for Fast Growth
(95% confidence level with a 16.1% confidence interval)

Reason	Answers	Percentage (%)
Active promotion	22	64.7
Trending technology	11	32.3
Innovative project	8	24.5
Active project	5	14.7
Project usability	2	5.8
Project usefulness	2	5.8
Unknown	2	5.8
Other	5	14.7

I believe the project has seen such great growth because of it's position within the greater Angular community ... (R87, Trending technology)

Table 12: Reasons for Viral Growth
(95% confidence level with a 18.8% confidence interval)

Reason	Answers	Percentage (%)
Promotion on social media sites	16	72.7
Code or documentation quality	6	27.2
Trending technology	4	19.0
Useful	3	14.2
New features	2	9.5
Other	2	9.5
Unknown	1	4.7

Summary: According to the surveyed developers, the major reason for *slow* growth is deprecation or lack of activity (53.8%). Regarding *moderate* growth, there are two conflicting sentiments on the developers' answers: positive sentiments (e.g., active promotion) and negative sentiments (e.g., niche audience). For *fast* growth, the three major reasons are active promotion, usage of trending technology, and innovative project. Finally, the major reason for *viral* growth is also promotion on social media sites (72.7%).

Implications for Empirical Software Engineering Researchers: The following observations are derived in this second survey regarding the selection of GitHub projects based on number of stars: (1) this selection might favor projects with successful marketing and advertising strategies, despite the adoption of well-established software engineering practices; (2) it is particularly important to check whether the projects have a viral growth behavior (e.g., CHRISLGARRY/APOLLO–11 gained 19,270 stars in just two weeks).

8. Threats to Validity

Dataset. GitHub has millions of repositories. We build our dataset by collecting the top-5,000 repositories by number of stars, which represents a small fraction in comparison to the GitHub's universe. However, our goal is exactly to investigate the most starred repositories. Furthermore, most GitHub repositories are forks and have very low activity [35–37].

Application domains. Because GitHub does not classify the repositories in domains, we performed this classification manually. Therefore, it is subjected to errors and inaccuracies. To mitigate this threat, the dubious classification decisions were discussed by the two paper's authors.

Survey study 1. The 5,000 repositories in our dataset have more than 21 million stars together. Despite this fact, we surveyed only the last developers who starred these repositories, a total of 4,370 developers. This decision was made to do not spam the developers. Moreover, we restricted the participants to those who gave a star in the last six months to increase the chances they remember the motivation for starring the projects. Another threat is related to the manual classification of the answers to derive the starring motivations. Although this activity has been done with special attention by the paper's first author, it is subjective by nature.

Survey study 2. In the second survey, we asked the developers to explain the reasons for the *slow*, *moderate*, *fast*, or *viral* growth observed in the number of stars of their repositories. For each growth pattern, we randomly selected a group of 100 repositories/developers. Exceptionally for repositories presenting a *viral* growth, 45 developers were used since they are the only ones with public e-mails. Since we received 115 answers (corresponding to a response ratio of 33.3%), we report the perceptions of a non-negligible number of developers.

Growth patterns. The selection of the number of clusters is a key parameter in algorithms like KSC. To mitigate this threat, we employed a heuristic that considers the intra/intercluster distance variations [23]. Furthermore, the analysis of growth patterns was based on the stars obtained in the last year. The stars before this period are not considered, since KSC requires time series with the same length.

Growth patterns characterization. In Section 6, we use a random forest classifier to identify the factors that distinguish the proposed growth patterns. This classifier requires the number of trees to compose a Random Forest. In this study, we used 100 trees, which is in the range suggested by Oshiro et. al. [38].

9. Related Work

We organize related work in four groups: (1) criteria for selecting GitHub projects; (2) studies on GitHub popularity metrics; (3) popularity of mobile apps; and (4) popularity of social media content.

Criteria for Selecting GitHub Projects: Stars are often used by researchers to select GitHub projects for empirical studies in software engineering [7–14]. For example, in a previous study, we use the top-5,000 GitHub repositories with most stars to investigate the performance of linear regression models to predict the number of stars in the future [39]. In a more recent study, we use the top-100 most starred GitHub repositories to investigate the channels used by open source project managers to promote their systems [40]. Ray et al. select 50 projects by stars on GitHub to study the effects of programming language features on defects [7]. To study the levels of participation of different open-source communities, Padhye et al. rely on the 89 most-starred GitHub projects [8]. Hilton et al. study Continuous Integration (CI) practices using a sample of 50 projects, ranked by number of stars [9]. Silva et al. select 748 Java projects to study refactoring practices among GitHub contributors [41] and Mazinanian study the adoption of lamba expressions in a large

sample of 2,000 Java projects, also ordered by stars [10]. Finally, Castro and Schots use GitHub stars as cut-off criterion to select projects and then analyze logging information to propose a visualization tool [14].

However, there are also studies that rely on different metrics and methodologies to select GitHub projects. For example, Vasilescu et al. use the programming language and number of forks to collect 246 repositories and then characterize the effects of CI in process automation on open source projects [42]. Bissyandé et al. use the default criteria of the GitHub API (i.e., best match) to collect 100K repositories and study popularity, interoperability, and impact of programming languages [43]. Kikas et al. combine number of issues and commits to obtain a large set of GitHub projects and study models to predict whether an issue will be closed [44].

Finally, there are efforts proposing more rigorous methods to select projecs in software repositories. For example, Falessi et al. first perform a systematic mapping study with 68 past studies and did not find any study that can be ranked as completely replicable [45]. Then, the authors present a rigorous method to select projects, called STRESS, that allows users to define the desired level of diversity, fit, and quality. Munaiah et al. propose a similar framework and tool, called Reaper, to select engineering GitHub projects, i.e., projects that follow sound software engineering practices, including documentation, testing, and project management [46]. Ultimately, Reaper was conceived to to separate the signal (e.g., engineering software projects) from the noise (e.g., home work assignments) when selecting projects in GitHub. As part of their findings, the authors report that using stars to classify engineered GitHub projects results on a very high precision, but with a low recall. In other words, repositories with a large number of stars are usually engineered projects; however, the contrary is not always true. Previously, Nagappan et al. proposed a measure, called sample coverage, to capture the percentage of projects in a population that are similar to a given sample [47]. Their goal is to promote the importance of diversity when selecting projects for evaluating a software engineering approach or performing an empirical study. They illustrated the usage of sample coverage in a population of 20K projects monitored by Ohloh.net, which is a public directory of open source projects, currently called Open Hub.

Studies on GitHub Popularity Metrics: Several studies investigate characteristics and usages of GitHub popularity metrics. Zho et al. study the frequency of folders used by 140K GitHub projects and their results suggest that the use of standard folders (e.g., doc, test, examples) may have an impact on project popularity, in terms of number of forks [48]. Aggarwal et al. study the effect of social interactions on GitHub projects' documentation [49]. They conclude that popular projects tend to attract more documentation collaborators. Jiang et al. provide a comprehensive analysis of inactive yet available assignees in popular GitHub projects. They show that some projects have more than 80% of inactive assignees [11]. Wanwangying et al. conduct a study to identify the most influential Python projects on GitHub [50]. They found that the most influential projects are not necessarily popular among GitHub users. By analyzing the effect of evolutionary software requirements on open source projects, Vlas et al. state that popularity (measured by number of stars and forks) depends on the continuous developing of requirements [51]. Papamichail et al. argue that the popularity of software components is as an indicator of software quality [52]; however, Herraiz et al. alert that popularity can also impact the perceived quality [53]. Finally, as one of the findings of a systematic mapping study, Cosentino et al. report that popularity (as measured by number of stars) is also useful to attract new developers to open source projects [37].

Popularity of mobile apps: Popularity in the context of mobile apps is the subject of several studies. For example, there are many studies examining the relationship between popularity of mobile apps and code

properties [27, 54–59]. Yuan et al. investigate 28 factors along eight dimensions to understand how high-rated Android applications are different from low-rated ones [27]. Their results show that external factors, like number of promotional images, are the most influential ones. Guerrouj and Baysal explore the relationships between mobile apps' success and API quality [60]. They found that changes and bugs in API methods are not strong predictors of apps' popularity. McIlroy et al. study the frequency of updates in popular free apps from different categories in the Google Play store [59]. They report that frequently-updated apps do not experience an increase in negative ratings by their users. Ruiz et al. examine the relationship between the number of ad libraries and app's user ratings [56]. Their results show that there is no relationship between these variables. Lee and Raghu tracked popular apps in the Apple Store and found that the survival rates of free apps are up to two times greater than the paid ones [57]. Moreover, they report that frequent feature updates can contribute to app survival among the top ones. Ali et al. conducted a comparative study of cross-platform apps to understand their characteristics [61]. They show that users can perceive and rate differently the same app on different platforms.

Popularity of social media content: Other studies track popularity on social networks, including video sharing sites (e.g., YouTube) and social platforms (e.g., Twitter and news aggregators). Chatzopoulou et al. [62] analyze popularity of YouTube videos by looking at properties and patterns metrics. They report that several popularity metrics are highly correlated. Lehmann et al. [21] analyze popularity peaks of hashtags. They found four usage patterns restricted to a two-week period centered on the peak time. Aniche et al. conduct a study to understand how developers use modern news aggregator sites (Reddit and Hacker News) [63]. According to their results, half of the participants read only the most upvoted comments and posts.

10. Conclusion

In this paper, we reported that developers star GitHub repositories due to three major reasons (which frequently overlap): to show appreciation to projects, to bookmark a project, and because they are using a project. Furthermore, three out of four developers declared they consider the number of stars before using or contributing to GitHub projects.

> *Recommendation #1:* Stars are a key metric about the evolution of GitHub projects; therefore, project managers should track and compare the number of stars of their projects with competitor ones.

We provided a quantitative characterization of the top-5,000 most starred repositories. We found that repositories owned by organizations have more stars than the ones owned by individuals (RQ #1). We also reported the existence of a moderate correlation of stars with contributors and forks, a low correlation between stars and commits, and no correlation between stars and repository' age (RQ #2). Furthermore, repositories have a tendency to receive more stars right after their public release (RQ #3). Finally, there is an acceleration in the number of stars gained after releases (RQ #4).

We validated the proposed stars growth patterns by means of a survey with project owners and core developers. We revealed that the major reason for a *slow* growth in the number of stars is project deprecation or inactivity. Regarding *moderate* growth, we detected both positive sentiments (e.g., active promotion) and negative ones (e.g., niche audience). The major reasons for *fast* growth are active promotion, usage of

trending technologies, and innovative projects. Finally, the major reason for *viral* growth is also promotion on social media.

> *Recommendation #2:* Open source projects require an investment on marketing and advertisement, mainly in social networks and programming forums, like Hacker News.

We distilled a list of threats practitioners and researchers may face when selecting GitHub projects based on the number of stars. For example, this selection favors large projects, with many contributors and forks. It may also include projects that receive a large number of stars in a short interval, including projects with a viral growth in their number of stars. Finally, it tends to favor projects with effective marketing and advertising strategies, which do not necessarily follow solid software engineering principles and practices.

> *Recommendation #3:* When selecting projects by number of stars, practitioners and researchers should check whether the stars are not concentrated in a short time period or whether they are mostly a consequence of active promotion in social media sites.

Future work may include an investigation of repositories that have few stars, including a comparison with the most starred ones. It would also be interesting to correlate repository's stars and language popularity and in this way to investigate relative measures of popularity. For example, if we restrict the analysis to a given language, a Scala repository can be considered more popular than a JavaScript one, although having less stars. Finally, the use of a different technique (e.g., Scott-Knott ESD [64]) may provide additional insights on the factors that impact the classification of a project in a given growth pattern.

> *Tool and dataset:* We implemented a tool to explore and check our results, including the time series of stars used in this paper and the proposed growth patterns. It is available at: `http://gittrends.io`. The analyzed data, manual classification of the application domain, and the surveyed responses used in this study are publicly available at: `https://doi.org/10.5281/zenodo.1183752`.

Acknowledgments

This research is supported by CAPES, CNPq, and FAPEMIG.

Bibliography

[1] G. Gousios, M. Pinzger, A. van Deursen, An exploratory study of the pull-based software development model, in: 36th International Conference on Software Engineering (ICSE), 2014, pp. 345–355.

[2] G. Gousios, A. Zaidman, M.-A. Storey, van Arie Deursen, Work practices and challenges in pull-based development: the integrator's perspective, in: 37th IEEE International Conference on Software Engineering (ICSE), 2015, pp. 358–368.

[3] Y. Yu, H. Wang, V. Filkov, P. Devanbu, B. Vasilescu, Wait for it: determinants of pull request evaluation latency on GitHub, in: 12th Working Conference on Mining Software Repositories (MSR), 2015, pp. 367–371.

[4] G. Gousios, M.-A. Storey, A. Bacchelli, Work practices and challenges in pull-based development: the contributor's perspective, in: 38th International Conference on Software Engineering (ICSE), 2016, pp. 1–12.

[5] A. Begel, J. Bosch, M. A. Storey, Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder, IEEE Software 30 (1) (2013) 52–66.

[6] B. Vasilescu, V. Filkov, A. Serebrenik, Stackoverflow and GitHub: Associations between software development and crowdsourced knowledge, in: 6th International Conference on Social Computing (SocialCom), IEEE, 2013, pp. 188–195.

[7] B. Ray, D. Posnett, V. Filkov, P. Devanbu, A large scale study of programming languages and code quality in GitHub, in: 22nd International Symposium on Foundations of Software Engineering (FSE), 2014, pp. 155–165.

[8] R. Padhye, S. Mani, V. S. Sinha, A study of external community contribution to open-source projects on GitHub, in: 11th Working Conference on Mining Software Repositories (MSR), 2014, pp. 332–335.

[9] M. Hilton, T. Tunnell, K. Huang, D. Marinov, D. Dig, Usage, costs, and benefits of continuous integration in open-source projects, in: 31st International Conference on Automated Software Engineering (ASE), 2016, pp. 426–437.

[10] D. Mazinanian, A. Ketkar, N. Tsantalis, D. Dig, Understanding the use of lambda expressions in Java, ACM on Programming Languages (PACMPL) 1 (OOPSLA) (2017) 85:1–85:31.

[11] J. Jiang, D. Lo, X. Ma, F. Feng, L. Zhang, Understanding inactive yet available assignees in GitHub, Information and Software Technology 91 (2017) 44–55.

[12] S. Nielebock, R. Heumüller, F. Ortmeier, Programmers do not favor lambda expressions for concurrent object-oriented code, Empirical Software Engineering.

[13] M. Rigger, S. Marr, S. Kell, D. Leopoldseder, H. Mössenböck, An analysis of x86-64 inline assembly in C programs, in: 14th International Conference on Virtual Execution Environments (VEE), 2018, pp. 84–99.

[14] D. Castro, M. Schots, Analysis of test log information through interactive visualizations, in: 26th International Conference on Program Comprehension (ICPC), 2018, pp. 1–11.

[15] H. Borges, A. Hora, M. T. Valente, Understanding the factors that impact the popularity of GitHub repositories, in: 32nd International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 1–11.

[16] G. Avelino, L. Passos, A. Hora, M. T. Valente, A novel approach for estimating truck factors, in: 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1–10.

[17] D. S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: 5th International Symposium on Empirical Software Engineering and Measurement (ESEM), 2011, pp. 275–284.

[18] D. E. Hinkle, W. Wiersma, S. G. Jurs, Applied Statistics for the Behavioral Sciences, 5th Edition, Houghton Mifflin, 2003.

[19] J. Yang, J. Leskovec, Patterns of temporal variation in online media, in: 4th International Conference on Web Search and Data Mining (WSDM), 2011, pp. 177–186.

[20] F. Figueiredo, On the prediction of popularity of trends and hits for user generated videos, in: 6th international Conference on Web Search and Data Mining (WSDM), 2013, pp. 741–746.

[21] J. Lehmann, G. alves, J. J. Ramasco, C. Cattuto, Dynamical classes of collective attention in Twitter, in: 21st International Conference on World Wide Web (WWW), 2012, pp. 251–260.

[22] J. A. Hartigan, Clustering algorithms, John Wiley & Sons, Inc., 1975.

[23] D. A. Menasce, V. Almeida, Capacity Planning for Web Services: Metrics, Models, and Methods, Prentice Hall, 2001.

[24] F. Figueiredo, J. M. Almeida, M. A. Gonçalves, F. Benevenuto, On the dynamics of social media popularity, ACM Transactions on Internet Technology 14 (4) (2014) 1–23.

[25] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[26] F. Provost, T. Fawcett, Robust classification for imprecise environments, Machine learning 42 (3) (2001) 203–231.

[27] Y. Tian, M. Nagappan, D. Lo, A. E. Hassan, What are the characteristics of high-rated apps? a case study on free Android applications, in: 31st International Conference on Software Maintenance and Evolution (ICSME), 2015, pp. 1–10.

[28] A. Hora, M. T. Valente, R. Robbes, N. Anquetil, When should internal interfaces be promoted to public?, in: 24th International Symposium on the Foundations of Software Engineering (FSE), 2016, pp. 280–291.

[29] J. Tsay, L. Dabbish, J. Herbsleb, Influence of social and technical factors for evaluating contribution in GitHub, in: 36th International Conference on Software Engineering (ICSE), 2014, pp. 356–366.

[30] M. S. Rakha, W. Shang, A. E. Hassan, Studying the needed effort for identifying duplicate issues, Empirical Software Engineering 21 (5) (2016) 1960–1989.

[31] E. Archer, rfPermute: Estimate Permutation p-Values for Random Forest Importance Metrics (2013). URL `https://CRAN.R-project.org/package=rfPermute`

[32] D. H. Wolpert, W. G. Macready, An efficient method to estimate bagging's generalization error, Machine Learning 35 (1) (1999) 41–55.

[33] J. Coelho, M. T. Valente, Why modern open source projects fail, in: 25th International Symposium on the Foundations of Software Engineering (FSE), 2017, pp. 186–196.

[34] R. Yates, B. Neto, et al., Modern information retrieval, ACM press New York, 1999.

[35] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, The promises and perils of mining GitHub, in: 11th Working Conference on Mining Software Repositories (MSR), 2014, pp. 92–101.

[36] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, An in-depth study of the promises and perils of mining GitHub, Empirical Software Engineering 21 (5) (2015) 1–37.

[37] V. Cosentino, J. L. C. Izquierdo, J. Cabot, A systematic mapping study of software development with GitHub, IEEE Access 5 (2017) 7173–7192.

[38] T. M. Oshiro, P. S. Perez, J. Baranauskas, How many trees in a random forest?, in: 8th International Workshop on Machine Learning and Data Mining in Pattern Recognition (MLDM), 2012, pp. 154–168.

[39] H. Borges, A. Hora, M. T. Valente, Predicting the popularity of GitHub repositories, in: 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE), 2016, p. 9.

[40] H. Borges, M. T. Valente, How do developers promote open source projects?, IEEE Computer 1 (1) (2018) 1–12.

[41] D. Silva, N. Tsantalis, M. T. Valente, Why we refactor? confessions of GitHub contributors, in: 24th International Symposium on the Foundations of Software Engineering (FSE), 2016, pp. 858–870.

[42] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub, in: 10th Joint Meeting on Foundations of Software Engineering (FSE), 2015, pp. 805–816.

[43] T. F. Bissyande, F. Thung, D. Lo, L. Jiang, L. Reveillere, Popularity, interoperability, and impact of programming languages in 100,000 open source projects, in: 37th Annual International Computer Software and Applications Conference (COMPSAC), 2013, pp. 303–312.

[44] R. Kikas, M. Dumas, D. Pfahl, Using dynamic and contextual features to predict issue lifetime in GitHub projects, in: 13th International Conference on Mining Software Repositories (MSR), 2016, pp. 291–302.

[45] D. Falessi, W. Smith, A. Serebrenik, Stress: A semi-automated, fully replicable approach for project selection, in: International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp. 151–156.

[46] N. Munaiah, S. Kroh, C. Cabrey, M. Nagappan, Curating GitHub for engineered software projects, Empirical Software Engineering 22 (6) (2017) 3219–3253.

[47] M. Nagappan, T. Zimmermann, C. Bird, Diversity in software engineering research, in: 9th Joint Meeting on Foundations of Software Engineering (FSE), 2013, pp. 466–476.

[48] J. Zhu, M. Zhou, A. Mockus, Patterns of folder use and project popularity: A case study of GitHub repositories, in: 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ISEM), 2014, pp. 1–4.

[49] K. Aggarwal, A. Hindle, E. Stroulia, Co-evolution of project documentation and popularity within GitHub, in: 11th Working Conference on Mining Software Repositories (MSR), 2014, pp. 360–363.

[50] W. Ma, L. Chen, Y. Zhou, B. Xu, What are the dominant projects in the GitHub Python ecosystem?, in: 3rd International Conference on Trustworthy Systems and their Applications (TSA), 2016, pp. 87–95.

[51] R. Vlas, W. Robinson, C. Vlas, Evolutionary software requirements factors and their effect on open source project attractiveness, in: 50th Hawaii International Conference on System Sciences (HICSS), 2017, pp. 1–11.

[52] M. Papamichail, T. Diamantopoulos, A. Symeonidis, User-perceived source code quality estimation based on static analysis metrics, in: 2nd International Conference on Software Quality, Reliability and Security (QRS), 2016, pp. 100–107.

[53] I. Herraiz, E. Shihab, T. H. Nguyen, A. E. Hassan, Impact of installation counts on perceived quality: A case study on Debian, in: 18th Working Conference on Reverse Engineering (WCRE), 2011, pp. 1–10.

[54] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, N. Sadeh, Why people hate your app: making sense of user feedback in a mobile app store, in: 19th International Conference on Knowledge Discovery and Data Mining (SIGKDD), 2013, pp. 1276–1284.

[55] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, D. Poshyvanyk, Api change and fault proneness: a threat to the success of Android apps, in: 9th Foundations of Software Engineering (FSE), 2013, pp. 477–487.

[56] I. J. Mojica Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, A. E. Hassan, Impact of ad libraries on ratings of Android mobile apps, IEEE Software 31 (6) (2014) 86–92.

[57] G. Lee, T. Raghu, Determinants of mobile apps' success: evidence from the app store market, Journal of Management Information Systems 31 (2) (2014) 133–170.

[58] L. Corral, I. Fronza, Better code for better apps: a study on source code quality and market success of Android applications, in: 2nd International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2015, pp. 22–32.

[59] S. McIlroy, N. Ali, A. E. Hassan, Fresh apps: an empirical study of frequently-updated mobile apps in the google play store, Empirical Software Engineering 21 (3) (2016) 1346–1370.

[60] L. Guerrouj, O. Baysal, Investigating the Android apps' success: an empirical study, in: 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1–4.

[61] M. Ali, M. E. Joorabchi, A. Mesbah, Same app, different app stores: A comparative study, in: 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2017, pp. 79–90.

[62] G. Chatzopoulou, C. Sheng, M. Faloutsos, A first step towards understanding popularity in YouTube, in: 30th IEEE International Conference on Computer Communications Workshops (INFOCOM), 2010, pp. 1–6.

[63] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, M. Gerosa, How modern news aggregators help development communities shape and share knowledge, in: 40th International Conference on Software Engineering (ICSE), 2018, pp. 1–12.

[64] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Transactions on Software Engineering 43 (1) (2017) 1–18.