NOVA IMS
INFORMATION MANAGEMENT SCHOOL

Master's Degree Program in
Information Management

Specialization in
Business Intelligence

To fork or not to fork, that's the GitHub question

What do successful forks have in common?

Pedro Gil Baptista Bernardino

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade NOVA de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

To fork or not to fork, that's the GitHub question

What do successful forks have in common?

by

Pedro Gil Baptista Bernardino

Master Thesis presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Business Intelligence

Supervised by

Flávio L. Pinheiro, NOVA Information Management School – Universidade Nova de Lisboa

November, 2025

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism, any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, November 2025

Pedro Gil Baptista Bernardino

ACKNOWLEDGEMENTS

I’d like to express my gratitude to Professor Flávio Pinheiro for continuously motivating me to keep on going with my thesis, whether through general guidance or staying until later during the thesis checkpoints to help me find the right direction to follow with the thesis, which at times I found myself lost.

I’d also like to thank my friends and family for continued motivation as well during the most difficult times when time seem to run out.

ABSTRACT

GitHub is a source code hosting website where users can host their code in repositories. Other users can then fork a repository which creates a copy of the original repository. Most forks are just made out of curiosity for a project, but others are developed way beyond the original repository and achieve renowned success. However, these are rather exception than rule which raises the question, what do successful forks have in common? Is it just context specific reasons unique to each project development, or are there common characteristics that can be identified from their metadata? We collected metadata from the 1000 most popular forks in GitHub and structured the forks chronological data into trees for all of them. These fork trees allowed us to analyze general statistics from the forks, as well as extract statistics from their evolution since their project inception. We were able to identify the forks from which fork tree structures were more likely to succeed, and obtained insights on some other factors that may or may not influence as much the success of a fork as initially expected.

KEYWORDS

GitHub; Repositories forks; Software popularity; Projects success; Innovation;

Sustainable Development Goals (SDG):

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

IT	Information Technology
KB	KiloBytes
MB	MegaBytes
SDT	Self Determination Theory

1 INTRODUCTION

GitHub is a website where its users can save the source code of their IT projects through the creation of repositories. The size of these repositories ranges from a single text file to enormous complex projects reliant on many different technologies and programming languages. GitHub also allows for any repositories that are set public for the internet, to be forked by creating an exact copy of the original repository files. Because of this, it means there are tens of millions of forks made by its users, with an average of five hundred thousand made daily.

The great majority of these forks don't differ from the original repository, as they were simply made to freely explore a project. Sometimes they are done to continue a project in a different direction from the original developers, while still retaining a strong connection to the original project. Other times it is to revive abandoned projects in stale repositories, therefore taking quasi new ownership of the old project through the creation of a fork. However, despite all these previous scenarios, no matter how good willed they are initially forked, the reality is that beyond the initial duplication process not many are able to achieve renown success. (Borges et al., 2016) That doesn't mean that in the millions of forked repositories, there aren't any truly distinguishable successful ones. Which brings us to our formulated research question (RQ):

RQ: What do successful forks have in common with each other?

In this work we try to answer this question by finding what characteristics successful forks have in common, and what might influence their rise to popularity by analyzing the repositories metadata. We sought to discover if metadata alone could reveal patterns of success.

The rest of this thesis is organized as follows: In ***Section 2***, we present a literature review of previous investigations that gave insights into our study. We hypothesize the motivation behind the creation of forks and gather knowledge of how reliable data in GitHub is.
In ***Section 3***, we describe the steps of the data collection process and provide some general statistics from the data collected.
In ***Section 4***, We describe the analysis conducted and present the results obtained. We discuss the conclusions reached and enumerate some limitations.
Finally in ***Section 5***, we conclude the work and present future work hypothesis.

2 LITERATURE REVIEW

2.1 BACKGROUND

While prior to 2015 there were still a couple competing alternative services for source code repositories such as Sourceforge and Bitbucket, it was the shutdown of Google Code in 2015 (“Bidding Farewell to Google Code,” 2015) that crowned GitHub as the most popular source code hosting service with over twenty one million repositories back then. Culminating with the acquisition by Microsoft in 2018. (Microsoft, 2018)

Since then, a plethora of studies have been conducted on GitHub about its dynamics and use cases, or using the near endless amount of repository data that it has been collecting since its inception in 2007. However, the nature of forks was never the main focus of previous studies. Even GitHub itself, of the several insights it generates about itself, forks aren’t one of them. (GitHub Innovation Graph, n.d.)

Before digging into the data, it begs to question in the first place, why are forks made?

2.2 PSYCHOLOGY OF FORKS

The diffusion theory claims that mass media interventions are best for raising awareness while interpersonal ones are needed to spur adoption. So, while many developers may discover a new project repository - and even stargaze it to further fuel its awareness - popularity alone likely isn’t enough to convince a lone actor to fork a project, and dedicate a reasonable amount of his time and effort to create something distinguishable enough from its origins. However, for example, if a group of original contributors of the project, who are familiar with each other, decide to split off and start a fork, they could support each other by balancing the workload and sharing knowledge, thus increasing the odds of success and making the fork a worthy endeavor to pursuit. As the diffusion theory claims that its strength lies in its explicit measure of the role of external influences and social networks in the adoption decision, the potential for a fork to happen could always exist, but there needs to be a spark (an external influence) for it to actually happen. Furthermore, it states that innovations flow through social networks which sometimes impede behavioral spread and sometimes accelerate it. In this context it could be interpreted as if a team of contributors in a project are happy with its direction, they see no reason to split off and go in a different direction. On the other hand, a project can become too stale or bureaucratic which stagnates its progress and innovation, thus incentivizing more revolutionary contributors to split off, and propel the project to new heights. (Diffusion of Innovations, n.d.)

The self-determination theory also attempts to explain the different kinds of motivations for developers to fork projects. It exemplifies intrinsic motivation as the inherent tendency to seek out novelty and challenges, to extend and exercise one's capacities, to explore, and to learn. Which is often the behavior of outsiders of a project that seek to learn about it, but not so much for insiders that already have the knowledge

advantage, that might prove crucial for a fork to succeed. A second sub theory of SDT is Organismic Integration Theory in which extrinsic motivation might be a stronger motivator behind successful forks, if not at worst by force. For example, a project suffered an internal rift for whatever reason. Some developers could decide to split off from the project and continue on their on accord, fulfilling their desires for self-control. Another motivator could be a stale project that started to fall behind on modern standards, and so some developers out of necessity take on the challenge of forking a project, just so it can become compliant with today's needs. As for the most part since GitHub projects are open source and voluntary, the motivations behind developers contributions are either a strong passion for a project, or a radical need for change. (Ryan & Deci, 2000)

Although lone actors could fork a project and continue on their own, another study on GitHub dynamics showed how programmers through stronger social interactions, learned programming languages more easily. In this specific case, the knowledge of programming languages was facilitated through cooperation between multiple people in GitHub projects. This learning vector could also be seen more abstractly as a means for knowledge to prosper, thus reassuring the importance of teamwork as a motivator behind prosperous forks. (Sanjay Guruprasad, 2018)

With some assumptions made as to what motivates developers to fork a project, we questioned what data would be most useful and revealing to conduct our analysis.

2.3 GitHub as a source of data for academic research

We searched for previous examples of analysis of GitHub data, to understand how to best analyze its data, and found several previous studies that analyzed different datasets.

Sanatinia & Noubir realized through projects purpose identification, that programming languages no matter their age and complexity, can be simply categorized between two general purposes, web programming and system oriented programming. (Sanatinia & Noubir, 2016)

Ray et al. identified using data from repository commits, that strong typed, static, functional programming languages appear to be less bug prone, thus resulting in better upfront code quality and less bug hunting. (Ray et al., 2017)

However, the main motivator behind our work was a previous study on factors that impact popularity of GitHub repositories. They found a strong positive correlation between stars and forks which intrigued us to pursue this study and find if indeed success breeds success. Another characteristic they found was that the majority of stars are received immediately or a week after a project goes public, so data collection most likely will find the forks that are already popular, not missing any in the making. (Borges et al., 2016)

However, another study pointed out that a project popularity relies more on in-code features than author metadata features, the latter is the kind of data we are looking to collect. (Weber & Luo, 2014)

Previous studies warn of the perils of mining data from GitHub, but their warnings are mostly directed towards self-identified metadata from randomized samples. (Kalliamvakou et al., 2014) That is data which, if available at all, was inserted by its own users which beyond the possibility of being error prone, could also be biased as to boast the public impression of their work. (Dabbish et al., 2012)

Data quality aside, can GitHub repositories stars even be a reliable metric of their true popularity? Despite the existence of online services that sell stars, just like one could buy followers for their Instragram page, (Nevado-Catalán et al., 2023) the legitimacy of stars themselves for the most part can safely be assumed as real, as GitHub frequently detects illegitimate stars by analyzing the legitimacy of users that star repositories *en masse.* (Aleksandra Sikora & Yassin Eldeeb, 2023) However, stars might not be most reliable metric for determining a repository's popularity, as they can mislead the judgement of the repository's popularity:

- The project can simply be very old, and had years to slowly accumulate a high quantity of stars;
- The project could have witnessed a golden age where it went viral and the star count suddenly surged in a small frame of time;
- The project could be very well optimized for search engines, making it very easy to find and thus able to easily reach many users that find it interesting, even if it wasn't what they were originally looking for.

Nevertheless, stars are still the most accessible intrinsic and public metric that can be used to measure a project's success. They are personal, singular and intentional. A star can only be given once by someone who intentionally wanted to demonstrate their interest in a project. Fake stars that could be given by automated fake accounts are largely detectable as previously stated. And unlike the metric of views that are used in other social networks, they are not automatically given just by visiting the repository page, they are manually given.

3 METHODOLOGY & DATA

To conduct our investigation on GitHub forks, we relied on data from fork repositories that had to be collected, prepared and analyzed as per the following steps:

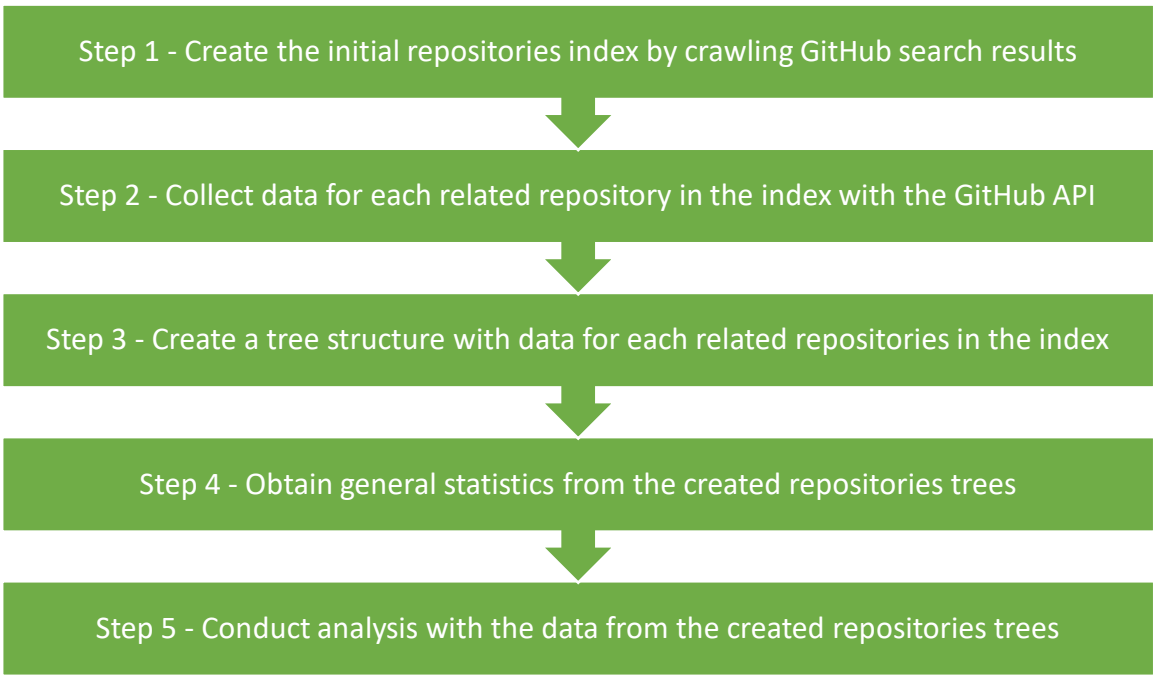


Figure 3-1 General steps conducted in our work.

We started our investigation by collecting metadata from existing GitHub forks in April 2025. To do that it was necessary to discover which GitHub repositories were forks from other repositories. GitHub search page provides parameters that allow us to filter the repositories search results to only return forks. Because of the huge number of forks available in GitHub, and because the large majority of them aren't different from their forked repositories, it was necessary to find the forks that would be analytically relevant to analyze. Due to lack of better repository metadata attributes, stars were deemed to be the best measure of recognizable success. We also wanted our sample data to be diverse enough to represent the variety of projects in GitHub. The parameter we used to increase the odds of finding unique and well-developed repositories, was by limiting our search to repositories with storage size larger than 250KB. This size was enough to discard smaller, less unique projects, but not too large that would only encompass too large and complex repositories. With these parameters defined we began building our repositories index by crawling the repositories names, from the search results pages with these parameters, starting from the most starred repositories to the least. We expanded our index until we reached the repositories count of 1000. This sample size was enough to represent the variety of repositories on GitHub, as from this point on, the frequency of repositories with a similar number of stars had already become too high, which would in turn reduce our sample diversity.

Following the indexation of the projects we proceeded to create a tree-like structure entitled “fork tree”, that would store the information belonging to each project related repositories, to allow for analysis of each fork chronological evolution and general statistics. This step was done as follows for each project:

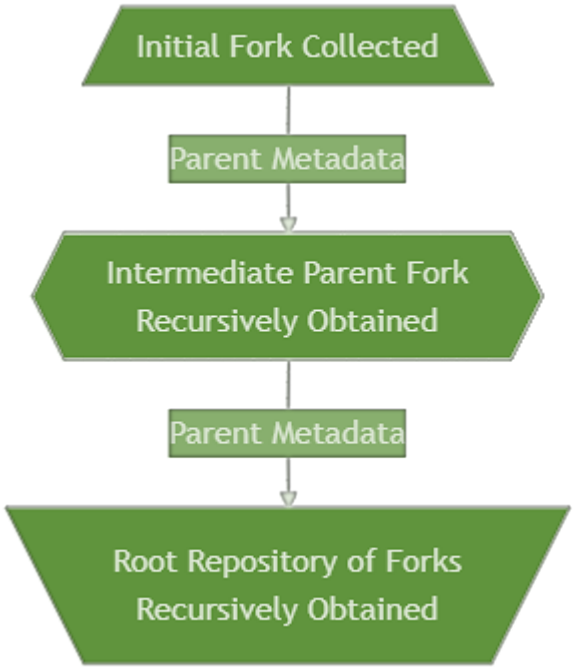


Figure 3-2 Abstract example of how a fork tree is constructed. There can be more than one intermediate fork between the root and the initial repository, or none at all.

Because we’ve indexed forks, the construction of the trees started from the chronological end of each project. As such, to reach the root repository of the project lineage, a regression in ramification levels was recursively done from last to first. At the bottom level sits the initial fork repository that was indexed. A level above could either be immediately found the root repository of the whole tree, or one or two intermediate fork repositories that were forked from the root repository of the tree. Each repository level except the root, had metadata with the name of the parent forked repository. Each level was recursively constructed until the root repository was reached. Despite the tree creation process having started from the bottom-up, we will interpret these fork trees from a top-down perspective. The initial indexed fork repository will be considered the last level of the tree, as per chronological order creation of the repositories within each tree.

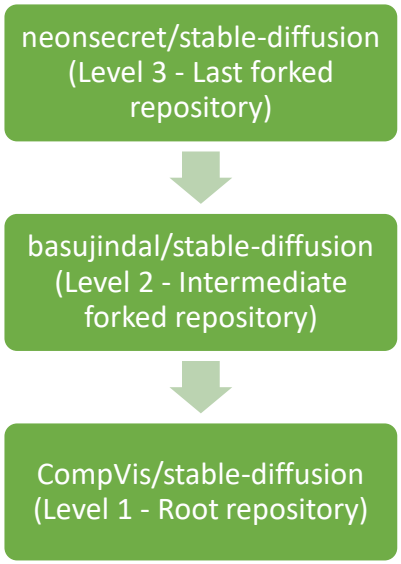


Figure 3-3 A real example of the construction of a fork tree.

The construction process seeded fork trees, each with a variable number of levels, and for each level we were able to collect the following fork repository metadata attributes:

Table 3-1 Original metadata collected from each repository

Attribute	**Description**	**Example**
Full Name	Complete name of repository structured as "CREATOR_NAME/REPOSITORY_NAME"	basujindal/stable-diffusion
Stargazers	Number of users who liked the repository.	54321
Watchers	Number of users following the repository.	12345
Language	The most common programming language found in the files from the repository.	Python
Size	Size of the repository project in kilobytes(KB).	1024
Parent	The *"Full Name"* of the parent repository that was forked from. For root repositories this attribute was empty.	CompVis/stable-diffusion
Archived	If the repository is marked as archived.	False

From the 1000 indexed project repositories, 997 fork trees were successfully seeded. Due to incomplete metadata, 3 projects were not able to complete their fork trees. As the incomplete projects only represented 0.3% of the sample data, the size of the remaining data sample was still deemed large enough to continue for analysis.

With the metadata from each level in the fork tree, custom attributes were organized to characterize the fork tree as a single homogenous entity. The following measures and metrics were extrapolated for analysis:

Table 3-2 Extrapolated data from each repository metadata

Data	Description
Root Name	The name of the top parent repository in the project's fork tree. Henceforth is named root repository.
Fork Name	The name of the latest descendant fork in the project's repositories tree.
Levels	How many repositories encompassed each project's repositories tree.
Language	The most common programming language found in the files from the repository. Certain projects had no language identified and were marked as "Unknown".
Created Date	The date the root repository was created.
Days Old	How many days old is the root repository until 1st of May 2025. (Circa date of data collection.)
Years Old	How many years old is the root repository until 1st of May 2025. (Circa date of data collection. Rounded down.)
Size of Root	The size of the files of the root repository in kilobytes.
Size of Last	The size of the files of the latest descendant fork in the project's repositories tree in kilobytes.
Size Average	The average size in kilobytes of all the repositories of each project's repositories tree.
Size Deviation	The standard deviation of size in kilobytes of all levels in the project's repositories tree.
Size Difference	The size difference in kilobytes between the root repository and the last fork repository in a project's repositories tree.
Size Biggest	The size in kilobytes of the biggest repository level in the project's repositories tree.

Average Days Between Forks	The average days between each fork creation in the project's repositories tree.
Days Between Forks Deviation	The standard deviation of the days between each fork creation in the project's repositories tree.
Total Stars	The sum of stars in all levels of the project's repositories tree.
Most Stars	The number of stars in the most popular repository of the project's repositories tree.
Most Starred Level	The level with the most stars in the project's repositories tree.
Is Last Level Most Starred	If the last level in the project's repositories tree is the most starred or not.
Stars Average	The average of stars of all levels in the project's repositories tree.
Stars Deviation	The standard deviation of stars of all levels in the project's repositories tree.
Stars Per Day Average	The average of stars gained per day of all levels in the project's repositories tree.
Total Watchers	The sum of watchers in all levels of the project's repositories tree.
Most Watchers	The number of watchers in the most popular repository of the project's repositories tree.
Most Watched Level	The level with the most watchers in the project's repositories tree.
Is Last Level Most Watched	If the last level in the project's repositories tree is the most watched or not.
Watchers Average	The average of watchers of all levels in the project's repositories tree.
Watchers Deviation	The standard deviation of watchers of all levels in the project's repositories tree.
Watchers Per Day Average	The average of watchers gained per day of all levels in the project's repositories tree.

Is Most Starred Level Same Most Watched Level	If the level in the project's repositories tree with the most stars is also the level with the most watchers.
Is Project Root Multiple Forked	If the fork's project tree root is the same as other fork trees in the data collected.

With all these attributes we produced global statistics from the sample data present in the fork trees:

Table 3-3 General statistics about fork trees levels and their sizes.

	Average Days Between Levels Creation	**Levels Average Size in MBytes**	**Size Difference Between Root and Last in MBytes**
mean	686.434	163.61	142.698
std	748.892	1559.65	1779.71
min	0	0.150391	0
25%	125	1.63184	0.358398
50%	431	7.89453	2.49219
75%	977	43.0522	21.2969
max	4283	37801.1	46596.2

Table 3-4 General statistics about fork trees stars and watchers.

	Stars Total	Highest Star Count	Stars Average	Stars Deviation	Watchers Total
mean	16401.4	15437.7	8076.33	7254.86	16401.4
std	40275.2	39974.6	20097.8	19854.8	40275.2
min	527	515	184	0.5	527
25%	1403	1007	673.5	336	1403
50%	3495	2558	1703	860.954	3495
75%	11332	9729	5571	4071.5	11332
max	356985	356447	178492	177954	356985

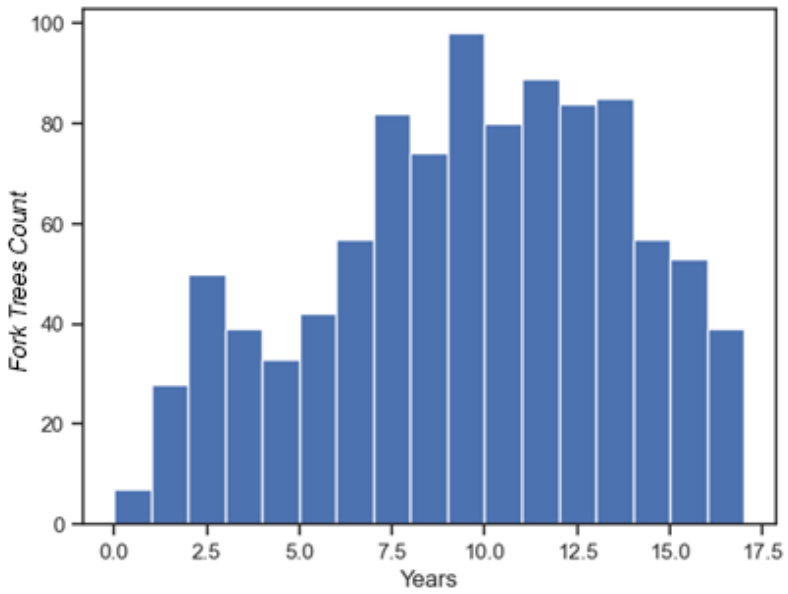


Figure 3-4 Distribution of age of fork trees projects in years.

In the Figure 3-4 we can see that most projects are between 7 and 14 years old.

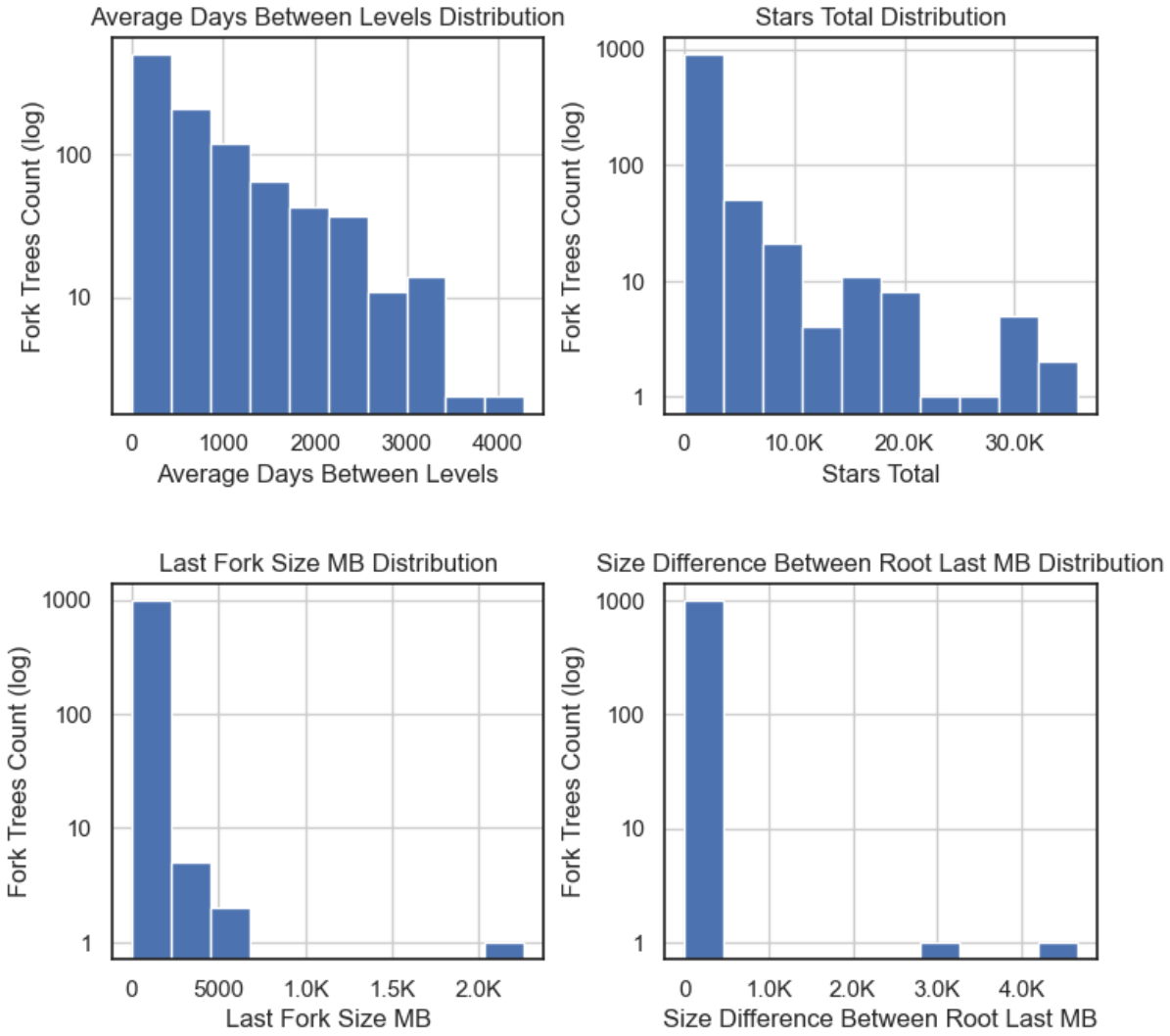


Figure 3-5 Distributions of several fork tree attributes.

Several distributions in Figure 3-5 also give us some insights into the scale of the data.

Table 3-5 Statistic about the archival status of the root repositories of fork trees.

Is Root Archived	**Total**
No	855
Yes	142

The status of archival of the root repository, could influence the chronological evolution of the fork tree popularity.

Table 3-6 Statistic of fork trees level count.

Levels	Total
2 Levels (root + last fork)	914
3 Levels (root + 1 middle fork + last fork)	68
4 Levels (root + 2 middle forks + last fork)	15

Over 90% of the forks were directly forked from the root repository. Less than 10% have an intermediate fork that separates them from the root repository. And barely 1% evolved twice until the last iteration.

Table 3-7 Statistic about levels in fork trees with most stars.

Levels With Most Stars	Total
Level 1 (Root)	552
Level 2	400
Level 3	35
Level 4	10

Over 50% of the fork trees, still had the root repository as the most starred one. As two-level fork trees were the most common kind, also explains why such a large quantity of second levels were the most starred. During data collection one project was discovered to have 5 levels in its fork tree, but was considered an outlier and was discarded, it was the source code of the video game Doom.

Table 3-8 Count of types of trees ramifications.

Trees Ramification Types	Total
Single (Only one popular fork lineage can be traced back to the root repository.)	847
Multiple (The root repository spans multiple popular fork derivatives.)	150

A phenomenon that was encountered was that a root repository could have forked into multiple successful fork repositories. In our sample, 60 projects were found to span 150 forks with at least a sibling with a root repository in common. For example, the repository CompVis/stable-diffusion was found to have 6 popular fork siblings. Ignoring the repeated root ramification trees, in total were identified 907 original projects.

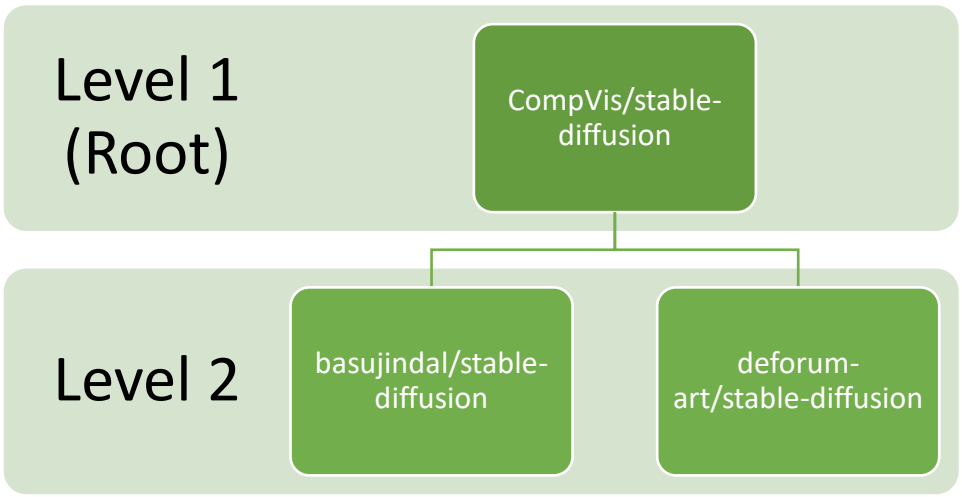


Figure 3-6 Example of a project root repository that produced multiple forks with distinct fork trees.

Table 3-9 Top 10 languages identified in the fork trees projects.

Languages	Total
Python	146
JavaScript	145
C++	90
C	88
Java	83
Unknown*	55
HTML	36
Go	36
TypeScript	33
C#	29

*No main language was identified in these projects' repositories.

From these general statistics we gained some initial insight into the data, to know where to start our research.

4 RESULTS AND DISCUSSION

With the objective of finding patterns for forks success, we've conducted analyses focused on two factors that have standout in the initial statistics, the number of levels in fork trees, and whether the fork tree root spanned multiple forks. Lastly we also conducted a brief analysis based on the languages of the projects but proved to be a dead end.

4.1 RESULTS

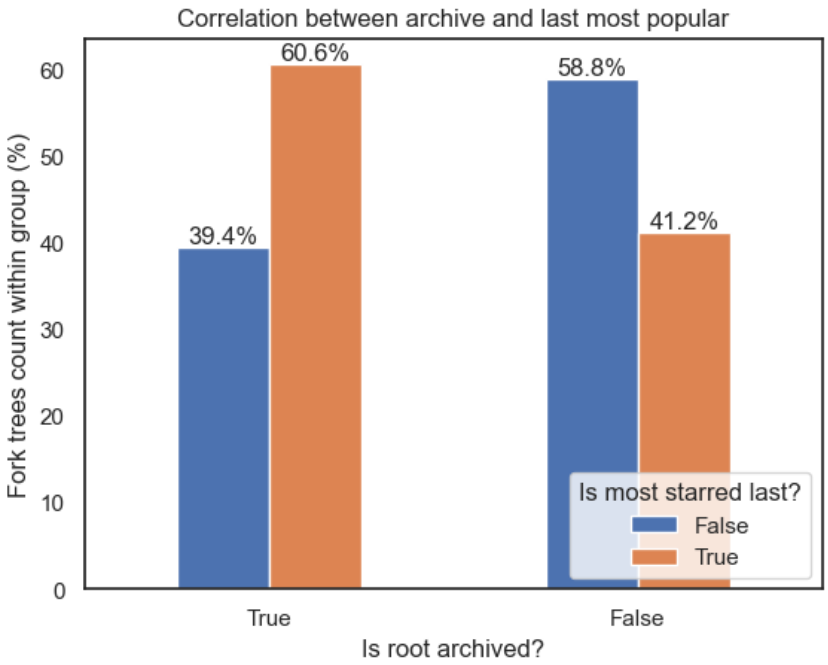


Figure 4-1 Influence of root repository archived status on the popularity of last fork.

We started our study with the hypothesis that a root repository with archived status could facilitate the popularization of the last fork. We found that the root repository was archived only in 142 of the 997 fork trees. Which judging by the results in Figure 4-1 the correlation is debatable. Although 60% is a reasonable percentage to be considered uncorrelated, we were expecting the difference between the fork trees with archived roots and those without, to have been stronger and more obvious. We were always expecting through sheer sample size for a reasonable number of forks to be able to surpass the root popularity, and indeed over 40% still managed to become the most starred.

We followed our popularity study by looking at how a root with multiple popular forks, could negatively influence the popularity of its offspring forks.

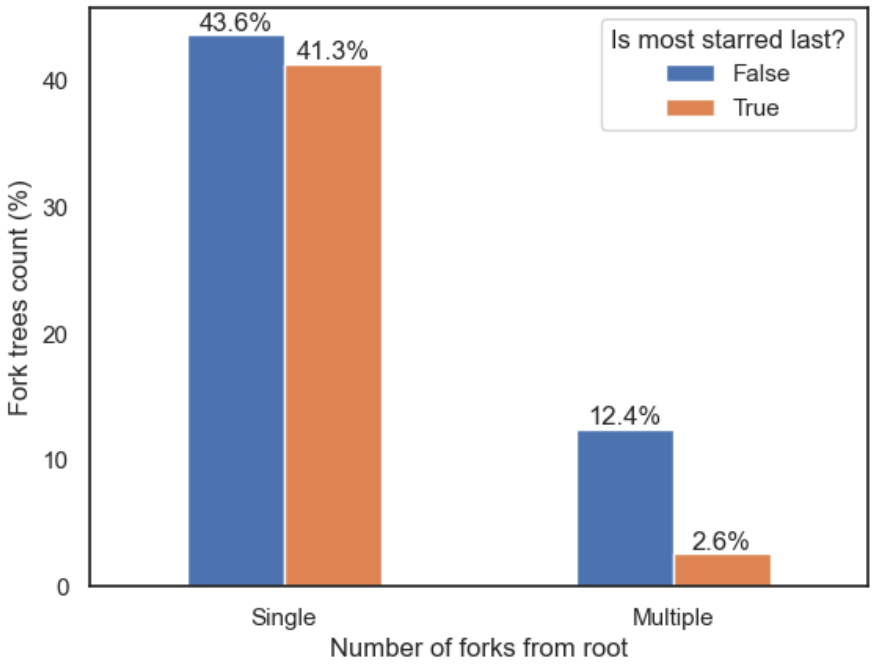


Figure 4-2 Comparison of last level most starred between fork tree ramifications.

Comparing the last fork popularity against fork ramifications in Figure 4-2 shows a distinguished pattern that was expected. While fork trees with a single fork lineage have a single repository to compete against its root or parent repositories, the ones where the root forked into multiple different forks, they struggle to successfully become more popular than the root as they likely compete between each other for popularity.

Following this clear trend, next we'll pivot this analysis to see how a multiple forked root could make it more difficult for fork trees to have more level depth.

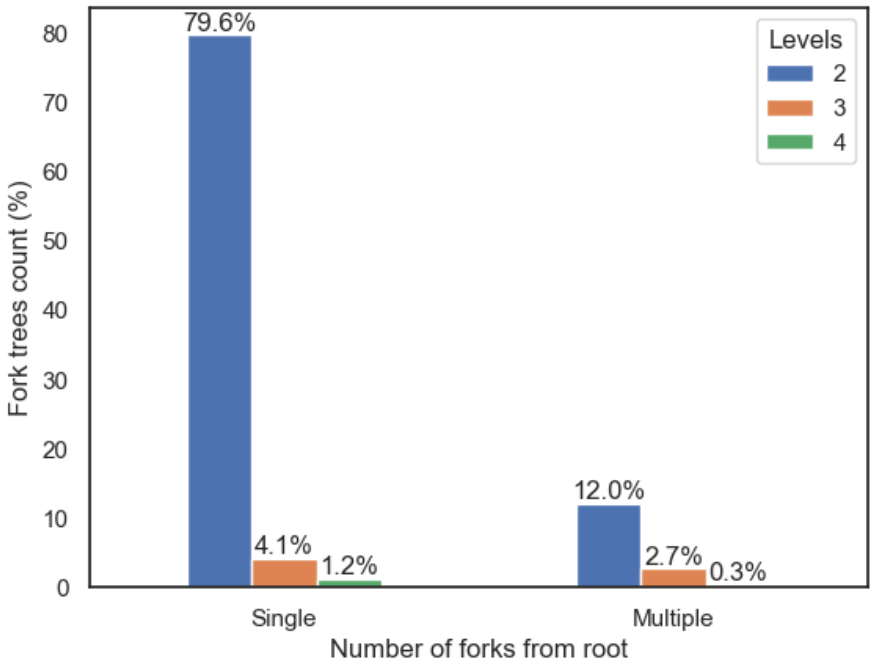


Figure 4-3 Distribution of fork trees between levels and fork tree ramifications.

Analyzing the levels differences between fork trees with and without multiple ramifications in Figure 4-3, shows just how big is the majority of projects with the simplest lineage. It also shows that ramifications with 3 and 4 levels don't have any proportionality to the amount of 2 levels fork trees. Further striking the difference between the two types of trees. While the number of simple lineage fork trees doesn't necessarily surprise, we were expecting a more linear or proportional decline of fork trees with 3 or 4 levels. After all, the development efforts would be focused on a single timeline, which we were expecting to make it more possible for further levels to exist, unlike the multiple forked roots.

However, we were still curious to see if wider root splits were less likely to have more levels than more concise forked roots.

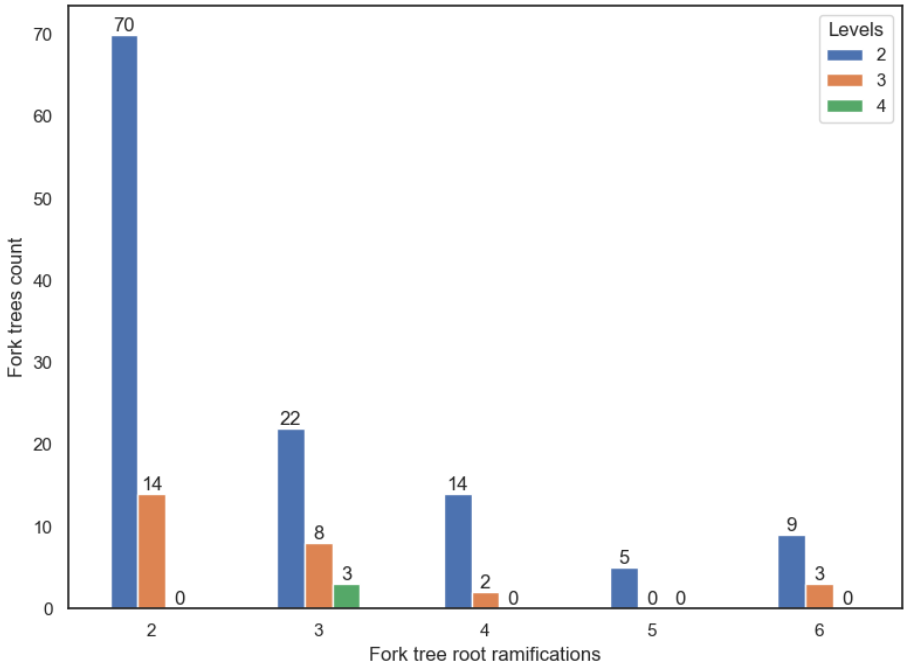


Figure 4-4 Distribution of levels for each multiplier of fork tree ramifications.

Milling down further the levels analysis for the multiple fork tree ramifications, reveals in Figure 4-4 different behaviors between all levels. 2 levels ramifications likely due to sheer number of projects, they end up quasi following a linear decline. In 3 levels ramifications the linear decline is much more discrete if at all acceptable given the V shape, which could be a sign of just randomness from the sample. Finally, 4 levels ramifications appear to be only found in very peculiar scenarios likely motivated by context specific situations. Overall, roots with up to 4 forks appear to show a linear decline the more forks one has. However, the 17 projects with 5 and 6 forks still show unexpected frequency for the amount of dispersion involved in these cases. Which might imply that such scenarios do not follow the linear trend, for context specific reasons. For example, it could be an organizational decision by the contributors of the root project, that decided to purposely split the project in multiple forks, instead of maintaining a single large root code base.

Having done the analysis from the ramifications' perspective, we took another look from a more general perspective.

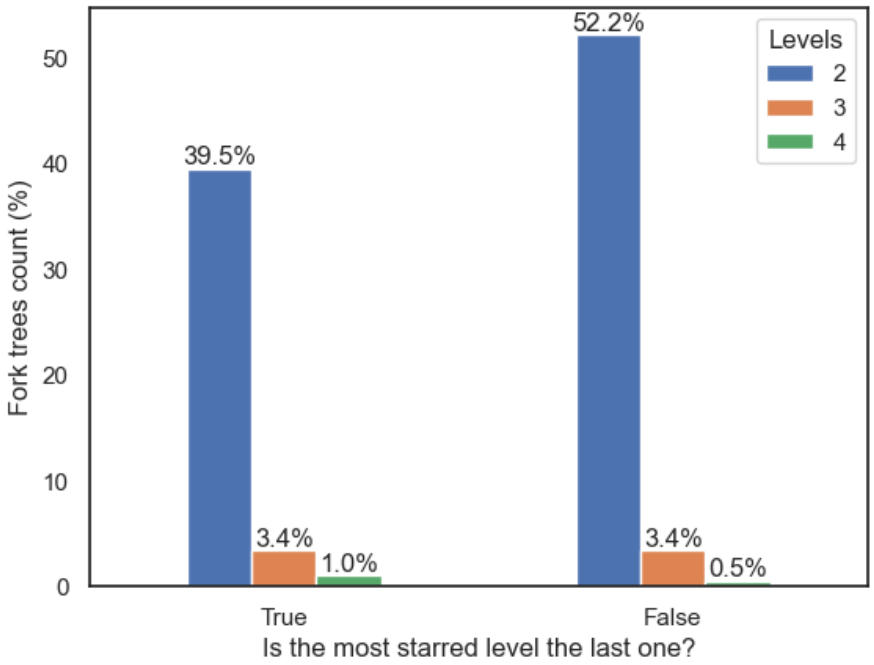


Figure 4-5 Percentage of fork trees levels where the last level is the most starred.

Analyzing the fork trees by which trees had the last level as the most starred in Figure 4-5, revealed that for the most part, despite many forks being sucessful in becoming more popular than their root repository, the majority still hasn't become more popular. Although curiously this difference is evenly balanced in fork trees with more than 2 levels. We weren't expecting the higher levels to be so evenly balanced, but the 2 wasn't so unexpected. Nevertheless, we wanted to find the reason behind the disparity in fork trees with 2 levels.

We followed with a sequence of analyses by comparing the age of fork trees where the last fork was the most popular versus the ones that wasn't.

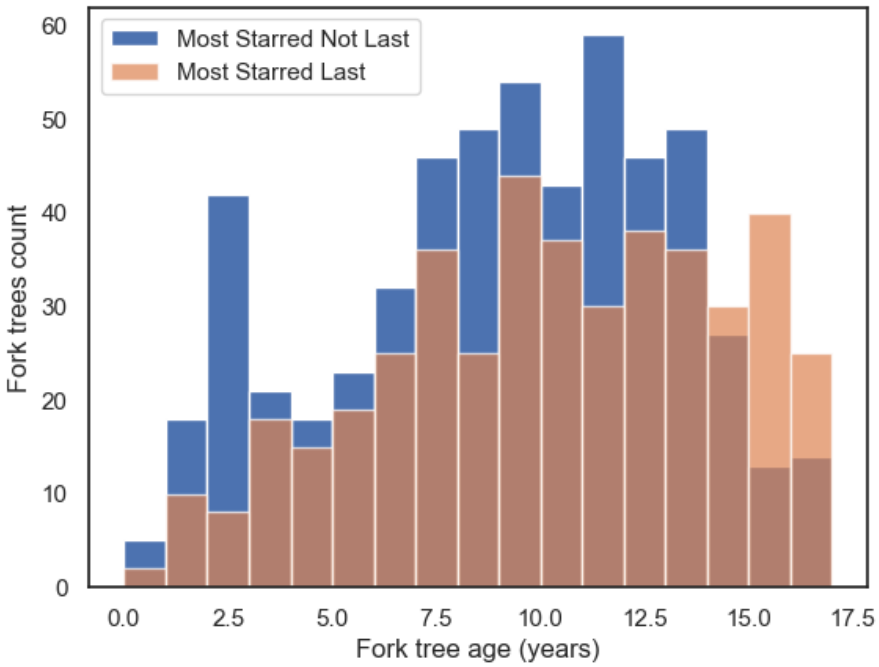


Figure 4-6 Histogram of fork trees where the last level is or is not the most starred.

The histogram Figure 4-6 appears to explain in part the disparity found in Figure 4-5. We can see it takes usually more than 3 years for a fork to become more popular than its root repository. But we can also see that given enough time, the latest forks sooner or later end up surprassing the previous forks of a project, as the older stops receiving stars in favor to the newest. We can see in fact, the majority of projects older than 14 years, the last fork is indeed the most popular of the fork tree. While in part not surprising, that it takes time for a fork to overtake its root acumulated stars, it is surprising by how long it takes.

However this could be due to many other factors, perhaps older projects in GitHub simply weren't very good in the first place, and contemporary forks were able to easily claim surpriority. Which appears to contradict the trend of the last 3 years where a clear majority of projects are still crowned by their root repository and not a popular fork. This could be due to lack of enough time for a fork to grow in popularity, or these modern projects established such a high standard that is hard to surpass.

Out of curiosity we drilled down the data on the fork trees older than 14 years, to see what kind of fork trees surpassed their roots popularity.

Table 4-1 Statistic of fork trees older than 14 years where the last level is the most starred by level count.

Levels	**Total**
2 Levels (root + last fork)	77
3 Levels (root + 1 middle fork + last fork)	14
4 Levels (root + 2 middle forks + last fork)	4

Table 4-2 Count of types of trees ramifications in fork trees older than 14 years where the last level is the most starred.

Trees Ramification Types	**Total**
Single (Only one popular fork lineage can be traced back to the root repository.)	90
Multiple (The root repository spans multiple popular fork derivatives.)	5

All that could be concluded from the drilled down data in Table 4-1 and Table 4-2 is that for the most part, even these older fork trees still follow the same patterns we've seen before in Figure 4-2 and Figure 4-5, the majority are single lineages with just 2 levels. That doesn't mean it isn't somewhat unexpected to still see some 3 level and 4 levels fork trees that managed to overtake the root repository popularity, given the dissipation of stars over the years and levels.

Finally, we decided to explore patterns in the data based on the main programming language detected on the fork tree code base.

Table 4-3 Fork trees levels distribution by repository main language

Language	2 levels (%)	3 levels (%)	4 levels (%)	Language Total
Python	88.36	8.90	2.74	146
JavaScript	92.41	6.90	0.69	145
C++	87.78	10.00	2.22	90
C	92.05	5.68	2.27	88
Java	92.77	7.23	0.00	83
Unknown	90.91	9.09	0.00	55
HTML	97.22	2.78	0.00	36
Go	94.44	2.78	2.78	36
TypeScript	96.97	3.03	0.00	33
C#	86.21	6.90	6.90	29
Jupyter Notebook	96.55	3.45	0.00	29
Shell	92.86	7.14	0.00	28
Ruby	85.19	11.11	3.70	27
Objective-C	73.08	23.08	3.85	26
PHP	96.00	4.00	0.00	25
CSS	100.00	0.00	0.00	16
Kotlin	93.33	6.67	0.00	15

In Table 4-3 we can see that 2 levels fork trees are the clear majority for most languages, and none stirred too much from this pattern. Almost all languages with less than 10 fork trees accounted had only 2 levels fork trees, which could be attributed due to too few developers with enough knowledge of the language capable of embracing a project and forking it. (Those languages are omitted from the table.)

Next, we conducted a similar analysis for languages but over roots with multiple forks.

Table 4-4 Fork trees ramifications distribution by repository main language

Language	Single (%)	Multiple (%)	Language Total
Python	88.36	11.64	146
JavaScript	85.52	14.48	145
C++	77.78	22.22	90
C	82.95	17.05	88
Java	87.95	12.05	83
Unknown	87.27	12.73	55
HTML	75.00	25.00	36
Go	94.44	5.56	36
TypeScript	90.91	9.09	33
C#	75.86	24.14	29
Jupyter Notebook	68.97	31.03	29
Shell	85.71	14.29	28
Ruby	96.30	3.70	27
Objective-C	80.77	19.23	26
PHP	92.00	8.00	25
CSS	75.00	25.00	16
Kotlin	73.33	26.67	15

A similar languages analysis was done regarding the ramification types of trees in Table 4-4, but a similar trend as previously noted was also found.

Overall, the programming languages identified in the fork tree root repository do not seem to reveal any unique patterns in between them. They all share the same patterns as found before, that the most common fork trees are single lineage with just 2 levels.

4.2 DISCUSSION

From the beginning the expectations were low to find any outstanding patterns from metadata alone. Especially when the reasons for projects to be forked are usually very context specific and not expected to be inferred from metadata alone, as was previously related in another study by Weber & Luo (2014).

Nevertheless, our results revealed two clear trends. First, most projects are only forked once, and its popularity is generally eventually passed onto that single fork repository.

Second, the projects that are forked multiple times fail to reach the popularity of the original project repository, as each sibling or level divides the attention that once concentrated on their parent repository project.

The reasons for this can't really be inferred from metadata alone. While a single fork somewhere in time is a reasonable endeavor that could result in success, for a project to survive multiple forks over the years must be very resilient with a unique history to justify its continued evolution. Just like how forks that have siblings struggle more to reach stardom, in comparison with forks that face no competition other than the parent repository they likely were created to replace. No matter how, projects with more than one popular fork, hardly reach the popularity of their roots. And even those who manage to overtake the root popularity often take a long time to do so, as Borges et al. (2016) realized when found that the majority of stars in repositories are earned when they go public, and then very slowly grow over time.

And while certain languages are more present in forked repositories, the patterns found between popular and less known languages are the same. So, the behavior for the forks development and creation appears to be the same no matter the language.

4.3 LIMITATIONS

There are some caveats worth nothing from the data collected:

First is that in reality there are many more forks beyond the ones officially tagged by GitHub. The metadata from the repositories collected still have embedded information of which repository they were forked from. However, any user can choose to cut loose its repository relation from the original repository, thus while technically no longer marked as a fork, in practice it is still a fork.

Second is that while most metadata collected is factual statistics, the language attribute is automatically determined by GitHub based on the majority of file types from a given repository. The reality is that most repositories have files from multiple programming languages, not just a single one.

Finally, third, although there was no clear correlation between root archival and fork popularity, we don't have information of when the root was archived. Forks with roots archived for longer periods of time could show a different pattern if the information was available.

5 CONCLUSIONS

To break the ice on the subject of GitHub forks was one of the main motivators behind our study, as they so often are mentioned in other studies, but rarely received a focused analysis. Although most project forks likely happen for context specific reasons that aren't noticeable in metadata, we still sought to find something new that might not have been so obvious. We've set on finding patterns of success by measuring popularity through repository stars, and comparing them with other fork attributes to see what could influence its stardom.

We started by collecting the metadata of the most starred fork repositories, and then recursively climbed its fork tree until we reached the forked root repository. Through each level climb we also collected the metadata of each repository. Finally, we assembled the fork tree structure with general chronological statistics of the project evolution. Once built it enabled practical continuous milling of the data for new insights as they were discovered on each iteration of the data.

We concluded that successfully forking a project once isn't too unlikely, but successive forks seem to increasingly struggle more to capture the same success as a direct fork from the root repository. Forks that also have many siblings that were split off their root repository, also appear to dissipate their popularity more, than if the project followed a single fork lineage. However, given enough time, no matter the level depth of the fork tree, or how split up the root repository was, all are set to achieve and surpass the popularity of their original repository, even if it takes years. On the contrary, the archival status of the root repository influence on the success of the last fork, wasn't as conclusive as we were hoping. The main programming language employed in the fork trees projects did not seem to influence the odds of fork success.

As for future research, although surveying the owners of fork repositories about the reasons why they forked might seem the easiest idea, a more hands on approach could be feeding the readme files of the repositories through a large language model, to analyze and try infer from the information, the reason why the fork was created. Another avenue that can be explored is to gather and analyze the collaborators data from the fork repositories, to try find a pattern or correlation between likelihood of forks, and the number of collaborators contributing to a forked project.

BIBLIOGRAPHICAL REFERENCES

Aleksandra Sikora & Yassin Eldeeb. (2023, June 1). *How Much Are GitHub Stars Worth to You?* The Guild. https://the-guild.dev/blog/judging-open-source-by-github-stars

Bidding farewell to Google Code. (2015). *Google Open Source Blog*. https://opensource.googleblog.com/2015/03/farewell-to-google-code.html

Borges, H., Hora, A., & Valente, M. T. (2016). Understanding the Factors that Impact the Popularity of GitHub Repositories. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 334–344. https://doi.org/10.1109/ICSME.2016.31

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, 1277–1286. https://doi.org/10.1145/2145204.2145396

Diffusion of innovations. (n.d.). Retrieved October 21, 2025, from https://www.nature.com/articles/gim200316.pdf

GitHub Innovation Graph. (n.d.). Retrieved October 27, 2025, from https://innovationgraph.github.com

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 92–101. https://doi.org/10.1145/2597073.2597074

Microsoft. (2018, June 4). Microsoft to acquire GitHub for $7.5 billion. *Microsoft to Acquire GitHub for $7.5 Billion*.

https://news.microsoft.com/source/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

Nevado-Catalán, D., Pastrana, S., Vallina-Rodriguez, N., & Tapiador, J. (2023). An analysis of fake social media engagement services. *Computers & Security*, *124*, 103013. https://doi.org/10.1016/j.cose.2022.103013

Ray, B., Posnett, D., Devanbu, P., & Filkov, V. (2017). A large-scale study of programming languages and code quality in GitHub. *Commun. ACM*, *60*(10), 91–100. https://doi.org/10.1145/3126905

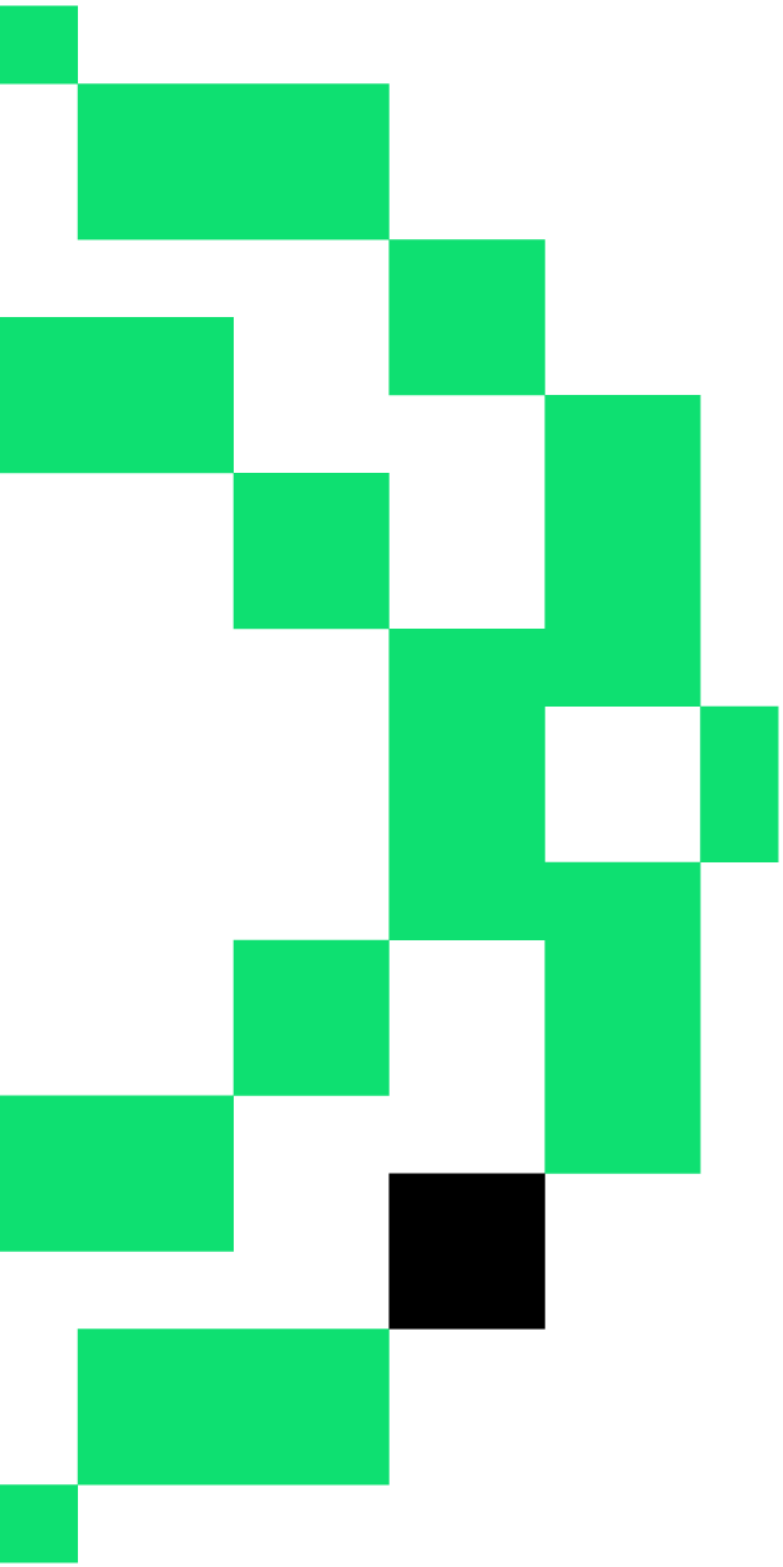
Ryan, R. M., & Deci, E. L. (2000). *Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being*.

Sanatinia, A., & Noubir, G. (2016). *On GitHub's Programming Languages* (No. arXiv:1603.00431). arXiv. https://doi.org/10.48550/arXiv.1603.00431

Sanjay Guruprasad. (2018). *The Influence of Collaboration Networks on Programming Language Acquisition*. https://dspace.mit.edu/bitstream/handle/1721.1/119085/1057897623-MIT.pdf?sequence=1&isAllowed=y

Weber, S., & Luo, J. (2014). What Makes an Open Source Code Popular on Git Hub? *2014 IEEE International Conference on Data Mining Workshop*, 851–855. https://doi.org/10.1109/ICDMW.2014.55

Data with Purpose.



Campus de Campolide, 1070-312 Lisboa, Portugal
novaims.unl.pt