CHARACTERIZING AND PREDICTING THE POPULARITY OF GITHUB PROJECTS

HUDSON SILVA BORGES

CHARACTERIZING AND PREDICTING THE POPULARITY OF GITHUB PROJECTS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Marco Túlio de Oliveira Valente

Belo Horizonte

Agosto de 2018

HUDSON SILVA BORGES

CHARACTERIZING AND PREDICTING THE POPULARITY OF GITHUB PROJECTS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Marco Túlio de Oliveira Valente

Belo Horizonte

August 2018

© 2018, Hudson Silva Borges.
Todos os direitos reservados.

Borges, Hudson Silva

B732c Characterizing and Predicting the Popularity of GitHub Projects / Hudson Silva Borges. — Belo Horizonte, 2018
xxii, 104 f. : il. ; 29cm

Tese (doutorado) — Federal University of Minas Gerais
Orientador: Marco Túlio de Oliveira Valente

1. Computação - Teses. 2. Open Source Software. 3. GitHub. 4. Popularity Growth Patterns. I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Characterizing and Predicting the Popularity of GitHub Projects

HUDSON SILVA BORGES

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCO TÚLIO DE OLIVEIRA VALENTE - Orientador
Departamento de Ciência da Computação - UFMG

PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - UFMG

PROF. FABRÍCIO BENEVENUTO DE SOUZA
Departamento de Ciência da Computação - UFMG

PROFA. INGRID OLIVEIRA DE NUNES
Departamento de Informática Aplicada - UFRGS

PROF. LEONARDO GRESTA PAULINO MURTA
Instituto de Computação - UFF

Belo Horizonte, 3 de setembro de 2018.

Agradecimentos

Gostaria de agradecer a todos que contribuiram, de forma direta ou indireta, para a conclusão desta tese de doutorado. Não seria possível concluir este trabalho sem o suporte de vocês. Eu agradeço a Deus por ter todas essas pessoas na minha vida.

Agradeço especialmente aos meus pais, Diva e Hugo, que sempre estiveram ao meu lado, apoiando minhas decisões e dando todo suporte necessário. Agradeço também a minha noiva, Elisângela, e sua família que estiveram comigo a todo momento.

Ao meu orientador, Professor Marco Tulio Valente, que nos últimos seis anos contribuiu de forma significativa para o meu crescimento pessoal e acadêmico.

Aos colegas do grupo de pesquisa ASERG pela amizade, pela agradável convivência diária e pela colaboração em diversos trabalhos.

Ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) por oferecer um curso de tal nível de excelência e o suporte financeiro e acadêmico.

Por fim, agradeço aos professores Eduardo Figueiredo, Fabrício Benevenuto, Ingrid Nunes e Leonardo Murta pela participação na minha defesa e pelas valiosas contribuições.

Abstract

Software popularity is a valuable information to modern open source developers, who constantly want to know whether their projects are attracting new users, whether new releases are gaining acceptance, or whether they are meeting user's expectations. However, we still have few studies about the popularity of open source projects. To tackle this problem, in this thesis, we propose and evaluate—through a set of quantitative and qualitative studies—practical guidelines and insights to help project managers understand and improve the popularity of their projects. First, we conduct a large-scale investigation with 791 developers to reveal their motivations for starring projects and to check whether they consider the number of stars before using or contributing to GitHub projects. The results reveal that three out four developers consider the number of stars and most developers star repositories to show appreciation and to bookmark projects. Next, we characterize the main factors that impact the number of stars of 5,000 GitHub projects. We reveal how popularity varies and correlates with other characteristics of a repository. We also investigate how often repositories lose popularity, how early they become popular, and what is the impact of new features in their popularity. Then, we propose, characterize, and validate a set of popularity growth patterns, which are derived after clustering the time series that describe the number of stars of the projects. After that, we address the most common channels used by developers to promote open source projects. We study these promotion channels and how often developers promote their projects. We also show how popular projects differ from random ones on the usage of promotion channels and what is the impact of promotion on news media sites. Finally, we use multiple linear regression models to predict the popularity of open source repositories. The results show that we can rely on the past six months to predict the number of stars six months ahead. Moreover, prediction models that consider the proposed growth patterns can reduce the average prediction error and reduce the amount of data needed to provide predictions.

Keywords: Open Source Development, Social Coding, Software Popularity.

Resumo

Popularidade é uma informação valiosa para desenvolvedores de projetos de código aberto, que querem constantemente saber se seus projetos estão atraindo novos usuários ou se novas *releases* estão atendendo às expectativas dos usuários. Contudo, poucos estudos investigam a popularidade de projetos de código aberto. Nesta tese, propomos e avaliamos—por meio de estudos quantitativos e qualitativos—um conjunto de diretrizes e informações práticas para ajudar gestores de projetos a entender e melhorar a popularidade de seus projetos. Primeiro, conduzimos uma investigação em larga escala com 791 desenvolvedores para revelar as motivações que os levam a "estrelar" um projeto e para verificar se eles consideram o número de estrelas antes de usarem ou contribuirem com projetos GitHub. Os resultados revelam que três em cada quatro desenvolvedores consideram o número de estrelas e a maioria "estrelam" repositórios para mostrar apreciação e para adicionar projetos em *bookmarks*. Em seguida, caracterizamos os principais fatores que afetam o número de estrelas de 5,000 projetos GitHub. Revelamos como a popularidade varia e se correlaciona com outras características dos repositórios. Também investigamos com que frequência os repositórios perdem popularidade, quão cedo eles se tornam populares e qual é o impacto de novas funcionalidades em sua popularidade. Em seguida, propomos, caracterizamos e validamos um conjunto de padrões de crescimento de popularidade, que são obtidos após clusterização das séries temporais que descrevem o número de estrelas dos projetos. Depois disso, investigamos os canais mais comuns usados pelos desenvolvedores para promover projetos de código aberto. Nós descrevemos os principais canais de promoção e analisamos a frequência com que desenvolvedores promovem seus projetos. Também mostramos como projetos populares diferem de outros projetos no uso dos canais de promoção e qual é o impacto da promoção em *sites* sociais de compartilhamento de notícias. Finalmente, usamos modelos de regressão linear múltipla para prever a popularidade de repositórios de código aberto. Os resultados mostram que podemos usar os dados dos últimos seis meses para prever o número de estrelas dos próximos seis meses. Além disso, modelos de previsão que consideram os padrões de crescimento propostos podem reduzir o erro médio e a quantidade de dados necessários para fornecer previsões.

Palavras-chave: Código Aberto, Codificação Social, Popularidade de Software.

List of Figures

1.1 Social coding features supported by GitHub (at the top of the figure) . . . 2
1.2 "Has Vue passed React yet?" site created to compare the number of stars of REACT and VUE.JS projects (`https://hasvuepassedreactyet.surge.sh`) 2
1.3 GitHub stars history of popular JavaScript-based MVC frameworks 3

2.1 Follow feature on GitHub 10
2.2 Forking feature on GitHub 11
2.3 Opening a pull request on GitHub 11
2.4 List of trending repositories on GitHub 12

3.1 How useful are the following metrics to assess the popularity of GitHub projects? (1: not useful; 4: very useful) 20
3.2 Age, number of commits, number of contributors, and number of forks (outliers are omitted) 23
3.3 Top-10 languages by number of repositories 24
3.4 Number of repositories by domain 25
3.5 Stars by programming language (considering only the top-10 languages with more repositories) 32
3.6 Popularity by application domain 33
3.7 Popularity by repository owner 33
3.8 Correlation analysis. In subfigures (c) and (d), the line is the identity relation 35
3.9 Number of stars and unstars events per repository (during April, 2018). The red line is the identity function. 37
3.10 Years between stars and unstars events by programming language. Outliers are omitted. 38
3.11 Years between stars and unstars events by domain. Outliers are omitted. . 39
3.12 Cumulative distribution of the time fraction a repository takes to receive 10%, 50%, and 90% of its stars 40

3.13 Fraction of stars gained in the first four weeks and in the last four weeks . 40
3.14 Fraction of stars for all releases (FS_{All}) and just after major releases (FS_{Major}) 41
3.15 REPORTR/DASHBOARD (the dots indicate weeks with releases) 42
3.16 Fraction of stars gained in the first week after all releases and just after the major releases 42
3.17 Fraction of stars gained in the week following all releases (or just the major releases) / fraction of time represented by these weeks 43
3.18 Fraction of stars by fraction of time (median values), computed using different time intervals 44
3.19 β_{CV} for $2 \leq k \leq 15$ 46
3.20 Clusters of time series produced by the KSC algorithm 47
3.21 Time series representing the centroids of each cluster 47
3.22 Examples of systems with viral growth 48
3.23 Rank differences in the interval of one year 49
3.24 Correlation analysis (as result, we removed features *a.pull_requests*, *a.contributors*, *r.network*, and *o.type*) 51

4.1 Number of GitHub stars of the analyzed projects 63
4.2 Most common promotion channels 64
4.3 Number of promotion channels per project 65
4.4 Distribution of the number of posts on the last 12 months (outliers are omitted) 66
4.5 Number of groups, cities, and countries of the user meetings 67
4.6 Most common promotion channels used by random projects 68
4.7 Number of posts, upvotes, and comments (outliers are omitted) 69
4.8 Number of GitHub stars received by projects covered by successful Hacker News posts in the first three days before and after the post publication . . 70

5.1 Cross Validation 75
5.2 β_{CV} $(2 \leq k \leq 15)$ 76
5.3 Five growth trends (clusters) identified for the repositories in our dataset . 76
5.4 Repositories popularity 79
5.5 JQUERY/JQUERY time series (369 weeks) 80
5.6 Generic model error 80
5.7 Stars vs RSE (generic model, $t_r = 26$ weeks) 81
5.8 Generic model error (y-axis). Predictions for 26, 52, and 104 weeks, using different fractions of data (x-axis) 83

5.9 Model prediction error for different growth trends (i.e., clusters extracted using the KSC algorithm) 84
5.10 Improvement of specific models per cluster (outliers are omitted). 84
5.11 Real vs predicted rankings. The red line is the identity function. 86
5.12 Spearman's rank correlation *rho* between predicted and real rankings per group of top-repositories (p-*value* < 0.001). 87

List of Tables

3.1 Descriptive statistics on the number of stars of the repositories in our dataset 22
3.2 Why do users star GitHub repositories? (95% confidence level with a 3.15% confidence interval) . . . 28
3.3 Do GitHub users consider the number of stars before using or contributing to a project? (95% confidence level with a 3.19% confidence interval) . . . 28
3.4 Features implemented in successful releases . . . 44
3.5 How the features are selected? . . . 44
3.6 Popularity Growth Patterns . . . 47
3.7 Factors potentially affecting the growth pattern of a repository . . . 50
3.8 Top-10 most influential factors (p-*value* < 0.01) . . . 52
3.9 Classification effectiveness . . . 53
3.10 Number of survey participants and answers per growth pattern – CI = Confidence interval at confidence level of 95% . . . 54
3.11 Reasons for Slow Growth (95% confidence level with a 19.1% confidence interval) . . . 55
3.12 Reasons for Moderate Growth – Positive Sentiments – (95% confidence level with a 16.9% confidence interval) . . . 56
3.13 Reasons for Moderate Growth – Negative Sentiments – (95% confidence level with a 16.9% confidence interval) . . . 56
3.14 Reasons for Fast Growth (95% confidence level with a 16.1% confidence interval) . . . 57
3.15 Reasons for Viral Growth (95% confidence level with a 18.8% confidence interval) . . . 57

4.1 Active Twitter, Facebook, and Google+ accounts . . . 66

5.1 Popularity Trends Description . . . 77
5.2 Top-10 repositories with more stars . . . 79

5.3 Number of stars gained (real and predicted measures) for the top-10 (first table half) and bottom-10 repositories (second table half). Predictions are produced using a generic model (t_r = 26 weeks and t = 52). We can see that the error (column "% Diff") is lower for the top-repositories. 82
5.4 Number of stars gained (real and predicted measures) for the top-10 (first table half) and bottom-10 repositories (second table half). Predictions are produced using specific models (t_r = 26 weeks and t = week 52). Column "% Improve" shows the gains achieved by specific models, when compared with the predictions provided by generic models. Black bars represent positive gains and gray bars denote negative gains. 85
5.5 Real and predicted rankings for the top-10 (first table half) and bottom-10 repositories (second table half), using the generic and the specific models. Marks "—" indicate repositories that were created and/or became popular after the date we set to select the repositories considered in this study. . . 88

Contents

Agradecimentos **ix**

Abstract **xi**

Resumo **xiii**

List of Figures **xv**

List of Tables **xix**

1 Introduction **1**
1.1 Problem and Motivation 1
1.2 Objectives 4
1.3 Contributions 4
1.4 Publications 7
1.5 Thesis Outline 7

2 Background and Related Work **9**
2.1 Social Coding and GitHub 9
2.2 Software Popularity 13
2.3 Popularity Prediction 15
2.4 Software Promotion 16
2.5 Concluding Remarks 17

3 Characterizing the Popularity of GitHub Projects **19**
3.1 Introduction 19
3.2 Dataset 21
3.3 Survey Study 25
3.3.1 Survey Design 25
3.3.2 Survey Results 26

3.4 Characterization Study . . . 30
3.4.1 Results . . . 31
3.5 Popularity Growth Patterns . . . 45
3.5.1 Proposed Growth Patterns . . . 46
3.6 Growth Patterns Characterization . . . 48
3.6.1 Methodology . . . 49
3.6.2 Most Influential Factors . . . 51
3.7 Developers' Perceptions on Growth Patterns . . . 53
3.7.1 Survey Design . . . 53
3.7.2 Survey Results . . . 54
3.8 Threats to Validity . . . 58
3.9 Conclusion . . . 59

4 Promotion on Open Source Projects **61**
4.1 Introduction . . . 61
4.2 Study Design . . . 62
4.3 Results . . . 64
4.4 Threats to Validity . . . 71
4.5 Concluding Remarks . . . 71

5 Predicting the Popularity of GitHub Repositories **73**
5.1 Introduction . . . 73
5.2 Study Design . . . 74
5.3 Dataset . . . 78
5.4 Results . . . 79
5.5 Threats to Validity . . . 89
5.6 Concluding Remarks . . . 89

6 Conclusion **91**
6.1 Summary . . . 91
6.2 Contributions . . . 93
6.3 Key Findings and Discussion . . . 95
6.4 Future Work . . . 96

Bibliography **97**

Chapter 1

Introduction

In this chapter, we start by stating our problem and motivation (Section 1.1). Next, we discuss the objectives, goals, and intended contributions of this thesis (Section 1.2). Then, we list our current publications (Section 1.4). Finally, we present the outline of this thesis (Section 1.5).

1.1 Problem and Motivation

Social coding platforms are disrupting the way developers collaborate on open source software development. For example, GitHub is the world's largest collection of open source software, with around 31 million users and 89 million repositories.[1] In addition to a `git`-based version control system, GitHub integrates features for social coding (as highlighted in Figure 1.1). 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 their changes in the main repository. Moreover, GitHub also supports features typical of modern social networks. For example, inspired by the *like* button of such networks, GitHub users can *star* a repository, presumably to manifest their interest or satisfaction with the hosted project [Begel et al., 2013].

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 various projects competing for the same users. For example, AngularJS, React, Vue.js, Backbone.js, and Ember.js compete for developers of single-page Web applications implemented in JavaScript. To illustrate

[1] `https://github.com/search`, verified on 06/26/2018.

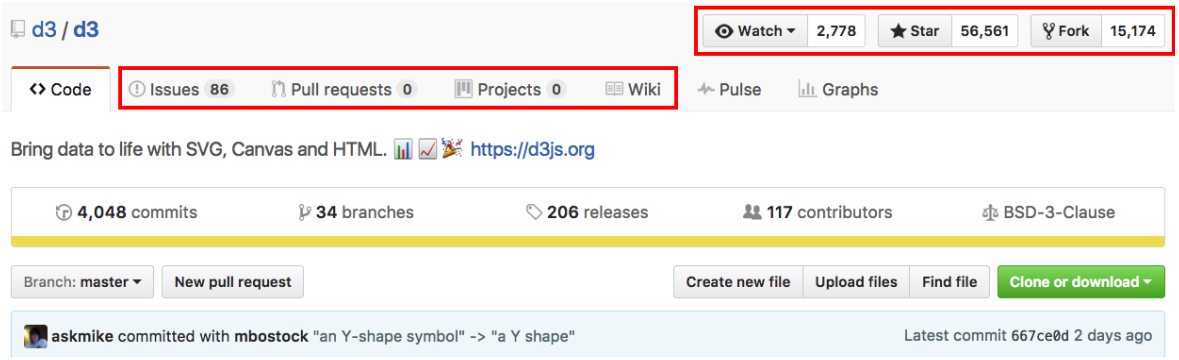


Figure 1.1: Social coding features supported by GitHub (at the top of the figure)

the relevance of stars for developers of these projects, in September 2016 the REACT project achieved the number of 50,000 GitHub stars. In order to celebrate this achievement the maintainers wrote a special blog post and created a commemorative t-shirt.[2] As a second example, in June 2018, VUE.JS surpassed REACT in number of stars and this fact was the subject of several discussions on news sites (e.g., `https://news.ycombinator.com/item?id=17316267`), social media sites (e.g., `https://twitter.com/dan_abramov/status/1007439168400654336`), and blogs (e.g., `https://zendev.com/2018/06/19/react-usage-beating-vue-angular.html`). In fact, a developer created a simple website to track the number of stars of both projects (Figure 1.2).

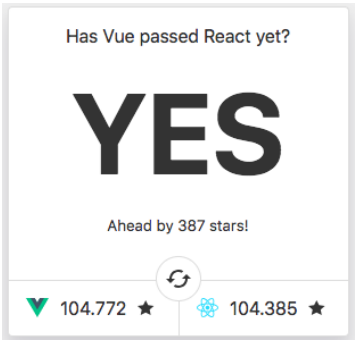


Figure 1.2: "Has Vue passed React yet?" site created to compare the number of stars of REACT and VUE.JS projects (`https://hasvuepassedreactyet.surge.sh`)

Figure 1.3 shows the GitHub stars history of ANGULARJS, REACT, VUE.JS, BACKBONE.JS, and EMBER.JS repositories. Created in January, 2010, ANGULAR/ANGULAR.JS is the oldest repository among these frameworks. However, the growing of its number of stars is decreasing since the creation of a new version of the framework, which is currently hosted on a new repository, called ANGULAR/ANGULAR.

[2] https://facebook.github.io/react/blog/2016/09/28/our-first-50000-stars.html

Few months later, jashkenas/backbone and emberjs/ember.js were created and, over time, they presented a similar growth in their number of stars. Maintained by Facebook, facebook/react quickly became popular, as showed by the high number of stars gained over time. Finally, vuejs/vue was created only five months later, gaining less stars until the end of 2015. After that, Vue.js started to gain more stars at the point to recently surpass facebook/react.

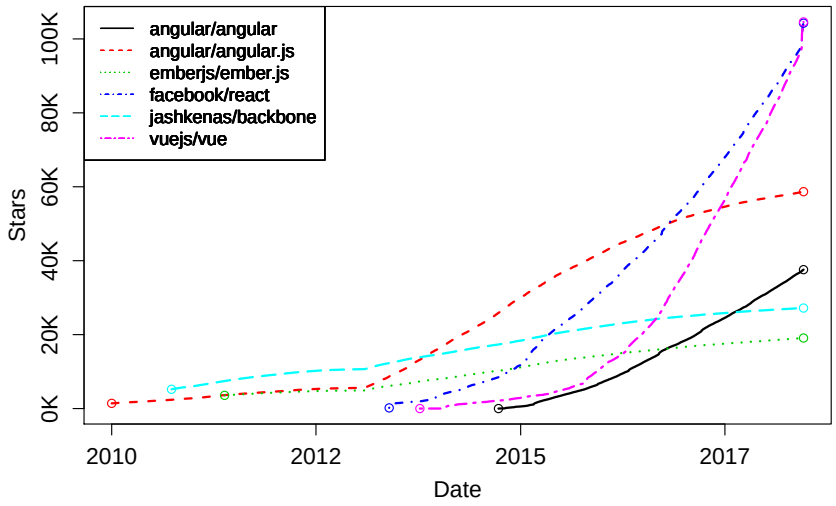


Figure 1.3: GitHub stars history of popular JavaScript-based MVC frameworks

However, several relevant questions about starring practices on GitHub remain opened. On the one hand, for project owners it is important to know the characteristics that make their frameworks popular, the different patterns in the growth of popularity, and which factors or events most influence the acceleration in the number of stars, as presented by Vue.js, for example. On the other hand, project users may have interest in technologies that will become popular in the future or even know if a given project is becoming stagnant in terms of popularity. Unfortunately, we have few studies about the popularity of open source projects. The exceptions are probably an attempt to distinguish popular and unpopular Python repositories using machine learning techniques [Weber and Luo, 2014] and a study on the effect of project's popularity on documentation quality [Aggarwal et al., 2014]. By contrast, popularity is extensively studied on other social platforms, like YouTube [Ahmed et al., 2013; Figueiredo et al., 2014] and Twitter [Lehmann et al., 2012; Ma et al., 2013]. These studies aim to guide content generators on producing successful social media content. Similarly, scientific knowledge on open source software popularity might also provide valuable insights on how to build and evolve systems in a competitive software market.

1.2 Objectives

This thesis has three main goals, as described next:

- Motivated by the lack of studies in the area, we intend to provide both a quantitative and qualitative characterization on the popularity of GitHub projects, as measured by their number of stars. By means of an analysis of the time series that describe the growth of the number of projects' stars, we intend to help developers to understand how popularity evolves on GitHub and the main factors that affect the popularity of their projects, including programming language, project domain, and frequent releases, among others. Additionally, by means of a set of surveys, we intend to collect developers' perceptions about the importance and meaning of GitHub stars. Ultimately, our goal is to assess the relevance of monitoring the number of stars of GitHub projects.

- We intend to provide practical guidelines and insights on how to improve the number of stars of GitHub projects. More specifically, based on the results of the aforementioned quantitative and qualitative studies, we concluded that promotion on social media sites is a important factor for gainning popularity on GitHub. Therefore, we intend to conduct surveys with developers to reveal the most common promotion channels they use to promote their projects.

- We intend to propose models to predict the popularity of GitHub projects. These models can be used by project owners to compare the popularity of their projects, in the future, with possible competitors. They can also help developers to support decisions on moving to or learning new technologies. Moreover, these predictions can benefit developers by providing a long-term vision. For example, they can prioritize projects with a consistent growing in favor of viral projects, which usually attract the developers' attention only for few days.

1.3 Contributions

The contributions achieved with this thesis are summarized as follows:

- **A quantitative characterization of the popularity of GitHub repositories**. To gain a first view of the popularity of GitHub projects, we investigate the main factors that impact the number of stars of a large set of 5,000 GitHub projects. This investigation aimed to answer the following research questions:

(RQ #1) How popularity varies per programming language, application domain, and repository owner? (RQ #2) Does popularity correlate with other characteristics of a repository, like age, number of commits, number of contributors, and number of forks? (RQ #3) How often do repositories lose popularity? (RQ #4) How early do repositories gain popularity? (RQ #5) What is the impact of new features on popularity? We found that JavaScript is the language with the highest number of popular repositories and the three domains whose repositories are more popular are systems software, applications, and web libraries and frameworks (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 a negligible correlation between stars and repository' age (RQ #2). We show that "unstar" events do not have a major influence on the popularity of GitHub repositories (RQ #3) and we concluded that repositories have a tendency to receive more stars right after their creation (RQ #4). Finally, we showed that there is an acceleration in the number of stars gained after releases (RQ #5).

- **A set of patterns to describe popularity growth**. We propose four patterns of popularity growth, which are derived after clustering the time series that describe the number of stars of GitHub projects. We show that almost 60% of the repositories present a slow growth in their number of stars and 30% present a moderate growth. Moreover, 9.3% of the repositories have a fast growth and only 2.3% present a massive (or viral) growth. We also show that the repository' age, number of issues, and last `git`-push are the most important features that distinguish repositories in the proposed growth patterns. The proposed popularity growth patterns can help project owners to reason about the popularity of their projects and also to compare this popularity with the one of competitors. Finally, project users can use the proposed patterns to prioritize projects with fast growth since they are more likely to receive contributions and to be maintained longer.

- **A qualitative characterization on the relevance of Github stars**. First, we present a large-scale investigation with 791 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. We found that GitHub developers star repositories mainly to show appreciation to projects (52.5%) and three out of four consider the number of stars before using or contributing to a project. Finally, we present a survey with 345 developers to reveal their perceptions on the proposed growth patterns. The results show that the major reason for *slow*

growth is deprecation or lack of activity and the major reason for *fast* and *viral* growth is promotion on social media sites (e.g., Hacker News).

- **A study on the channels used to promote open source projects**. The use of promotion channels can help to keep the interest of the community and also to attract new members. In this study, we address the most common channels used by developers to promote open source projects. We ask four research questions: (RQ #1) What are the most common promotion channels? (RQ #2) How often do developers promote their projects? (RQ #3) How popular and random projects differ on the usage of promotion channels? and (RQ #4) What is the impact of promotion on Hacker News? We show that Twitter and User Meetings are the most common channels whereas Facebook and Google+ are the less common ones (RQ #1). We also show that most projects that use blogs post more than once a month (RQ #2). More importantly, we found that the usage of promotion channels is significantly lower among random projects compared to the popular ones (RQ #3). Finally, we show that promotion on Hacker News (a popular news aggregator site[3]) has a major impact on the number of stars of the projects (RQ #4).

- **A model to predict the popularity of GitHub repositories**. Repository owners and clients usually want to know how their projects will perform in the future, when compared with competitor projects. In this final study, we use multiple linear regression models to predict the popularity of GitHub repositories. Specifically, we compute and apply such models over two types of data: generic and specific. By generic, we refer to models produced using the top GitHub repositories, including 4,248 popular GitHub repositories. By specific, we refer to models produced from repositories that share similar growth patterns. We evaluate the proposed models by asking three major research questions: (RQ #1) What is the accuracy of the generic prediction models? (RQ #2) What is the accuracy of the specific prediction models? (RQ #3) What is the accuracy of the repositories' rank as predicted using the generic and specific models? We found that general models start to provide accurate predictions when they are trained with data from six months and used to predict the number of stars six months ahead (RQ #1). We also found that specific models can reduce the average prediction error (e.g., 15% on median for slow growth pattern); they also require less data to provide predictions, e.g., 10 weeks for slow growth pattern

[3] `https://news.ycombinator.com`, verified on 06/26/2018.

(RQ #2). Finally, we report a very strong correlation between predicted and real rankings (RQ #3). In summary, the proposed predictions models can help both project owners and users. For example, project owners can use the predictions to support long-term decisions whereas users can use the predictions to support their decisions on moving to or adopting new software technologies.

1.4 Publications

The following publications document the results we have accomplished during this Ph.D work:

- Hudson Borges, Andre Hora, Marco Tulio Valente. Understanding the Factors that Impact the Popularity of GitHub Repositories. In 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 1-11, 2016. Qualis A1.

- Hudson Borges, Andre Hora, Marco Tulio Valente. Predicting the Popularity of GitHub Repositories. In 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE), p. 1-10, 2016. Qualis B2.

- Hudson Borges, Jailton Coelho, Paulo Carvalho, Mariana Fernandes, Marco Tulio Valente. Como Pesquisadores Usam o Dataset GHTorrent?. In 5th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM), p. 1-8, 2017. Qualis B5.

- Hudson Borges and Marco Tulio Valente. How do Developers Promote Open Source Projects?. Accepted for IEEE Computer, 2018. Qualis A1, JCR 2018: 1.98.

- Hudson Borges and Marco Tulio Valente. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. Journal of Systems and Software, vol. 146, p. 112-129, 2018. Qualis A2, JCR 2018: 2.27.

1.5 Thesis Outline

This thesis is structured in the following chapters:

- Chapter 2 describes background on social coding, software popularity, data prediction, and channels commonly used to promote open source projects.

- Chapter 3 presents an in-depth investigation on the popularity of GitHub repositories. First, we reveal the developers' motivations for starring projects on GitHub. Then, we study the main factors that impact the number of stars of GitHub projects and investigate the impact of new features on project popularity. Finally, we identify, characterize, and explore the developers' perceptions on four main popularity growth patterns.
- Chapter 4 explores the most common channels used by developers to promote open source projects.
- Chapter 5 investigates the use of multiple linear regression models to predict the number of stars of GitHub repositories.
- Chapter 6 concludes this thesis and outlines future work ideas.

Chapter 2

Background and Related Work

In this chapter, we present background information and work related to this PhD thesis. First, we introduce the concept of social coding and discuss how GitHub, the most popular social coding platform nowadays, implements social features; we also present studies that investigate the impact of such features on software development (Section 2.1). Next, in Section 2.2 we present studies related to software popularity in app stores and social media sites. In Section 2.4, we discuss studies related to the promotion of open source projects. Moreover, in Section 2.3 we present an overview on the applications of data prediction techniques in different contexts of software engineering. Finally, we conclude this chapter with general remarks on the discussed topics (Section 2.5).

2.1 Social Coding and GitHub

Over the last years, open source software development has become more social and collaborative. In fact, social coding emphasizes formal and informal collaboration in the software development by empowering knowledge exchange between the developers. In this context, traditional code management tools (e.g., control version systems, issue tracking, etc) are combined with social networks features to connect developers and increase their social interactions. Among the platforms that support social coding, GitHub is nowadays the most popular one and the world's largest collection of open source software, with around 31 million users and 89 million repositories.[1]

Due to its vast number of social features, GitHub has been the major data source for several studies in software engineering. For example, on GitHub, developers can

[1] `https://github.com/search`, verified on 06/26/2018

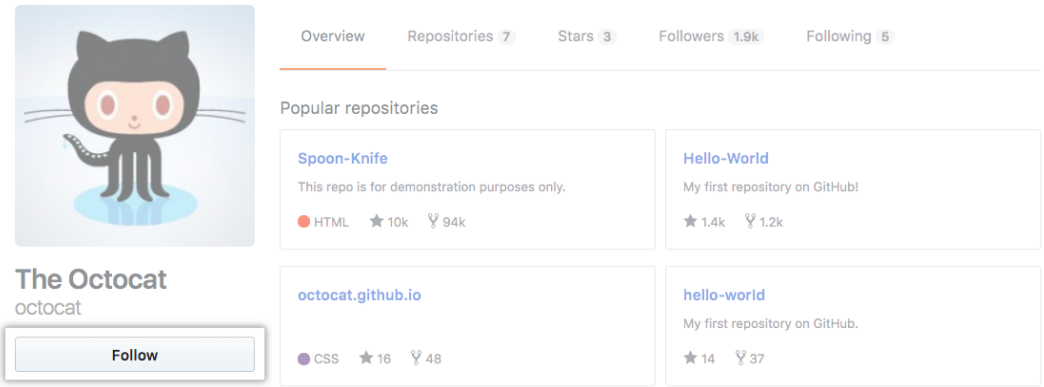


Figure 2.1: Follow feature on GitHub

follow other developers to listen their activities on the platform (Figure 2.1). These social interactions and the implications on the software development are the subject of several studies [Jiang et al., 2013; Wu et al., 2014; Blincoe et al., 2016]. Jiang et al. [2013] conduct a study to understand the influence of developers connections on project dissemination in GitHub. To this purpose, the authors collected more than 2 million social links from 747 thousand users. Then, the authors construct social graphs and characterize the role played by these links. They show that social links play a notable role by helping in projects dissemination and also by influencing developers to participate in projects. Wu et al. [2014] analyze to what extend the *following* feature on GitHub is similar to features in other social media platforms. Particularly, they examined developers' interactions inside and outside the HOMEBREW project. According to their findings, most of the *follow* activities in this project are not product of developers' collaborations on GitHub, but from interactions on other social platforms (i.e., Twitter, HackerNews, StackOverflow, etc). Finally, Blincoe et al. [2016] investigate why GitHub users make use of the *follow* feature and the influence of the most *followed* users on their *followers*. To identify the motivations behind following other developers, the authors surveyed 800 GitHub users. They found that receiving updates on activities, discovering new projects and trends, and learning are some of the reasons for following popular users. Moreover, they also reported that followers are likely to contribute to new projects after a popular user whom they are following performs an activity on that project.

Forking is another widely used feature provided by GitHub. By forking a repository, developers can create a copy of the project in their own account to freely modify, without affecting the original project. It is also important to observe that the copy can be made by anyone without requiring the permission of the repository's owner

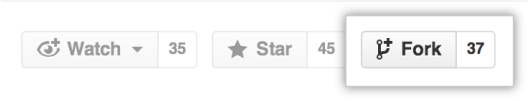


Figure 2.2: Forking feature on GitHub

(Figure 2.2). Jiang et al. [2016] analyze the characteristics of forking behavior by investigating almost 2 million GitHub forks. Their findings show that developers fork repositories mainly to submit pull requests, fix bugs, add new features, and keep copies. Moreover, developers are likely to fork projects implemented in their preferred programming languages and mostly forks are copied from the original owner (i.e., developers usually do not fork from other forked repositories). Finally, Zhou et al. [2018] present an approach to automatically identify and label features among code changes in forks and to provide a compact overview of features and their implementations. To this purpose, they cluster the changed code relationships and their dependencies to label each identified feature with representative keywords extracted from commit messages, code, and comments. According to the authors, their technique produces more accurate labels than previous ones.

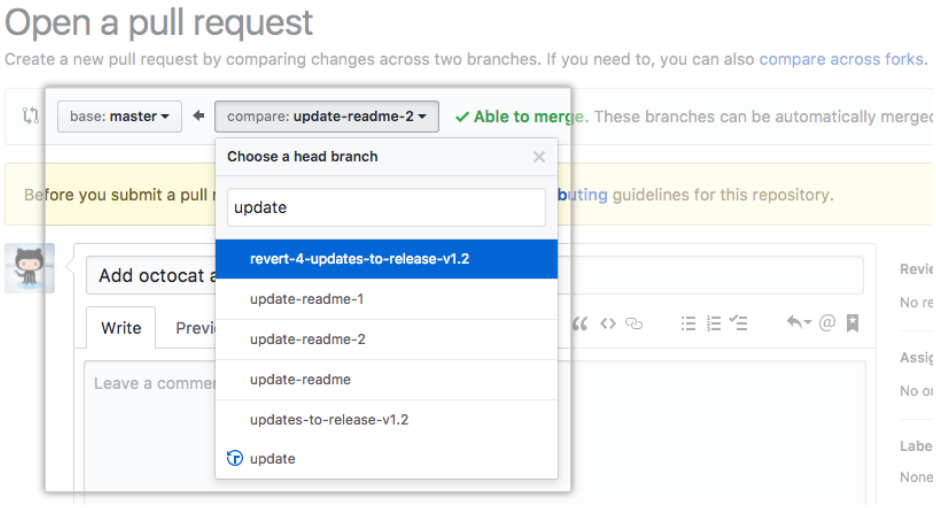


Figure 2.3: Opening a pull request on GitHub

According to Jiang et al. [2016], the major reason for forking repositories is to submit pull requests. As illustrated in Figure 2.3, pull request is a feature that allow developers, who are not members of a given project, to submit their changes (e.g., bug fixes, new features, or code improvements) to the original project. After the submission, the request is analyzed by developers of the original repository and, if accepted, merged to the source code. The pull-based software development model was explored in other studies [Gousios et al., 2014; Yu et al., 2015; Gousios et al., 2015]. Gousios et al. [2014]

explore how pull-based development works by analyzing almost 2 million pull requests and then identifying the factors that affect pull request lifetime, merging, and rejection. They show that the pull-based model is used by only 14% of the analyzed projects, and most of the pull requests affect just a few lines of code. In another study, Gousios et al. [2015] investigate the work practices and the challenges that integrators face while working in pull-based settings. By means of a qualitative investigation and survey with integrators, they found that integrators struggle to maintain the quality of their projects and have difficulties with prioritizing contributions that need to be merged. At last, Yu et al. [2015] report an investigation on which factors affect pull request evaluation latency in GitHub. To this purpose, they used a multiple linear regression to model the latency of pull requests evaluation. Their results show that small changes on pull requests (e.g., low number of lines modified) increases the reviewing chances and that number of comments is the best latency predictor (i.e., pull requests with more comments are likely to be resolved faster).

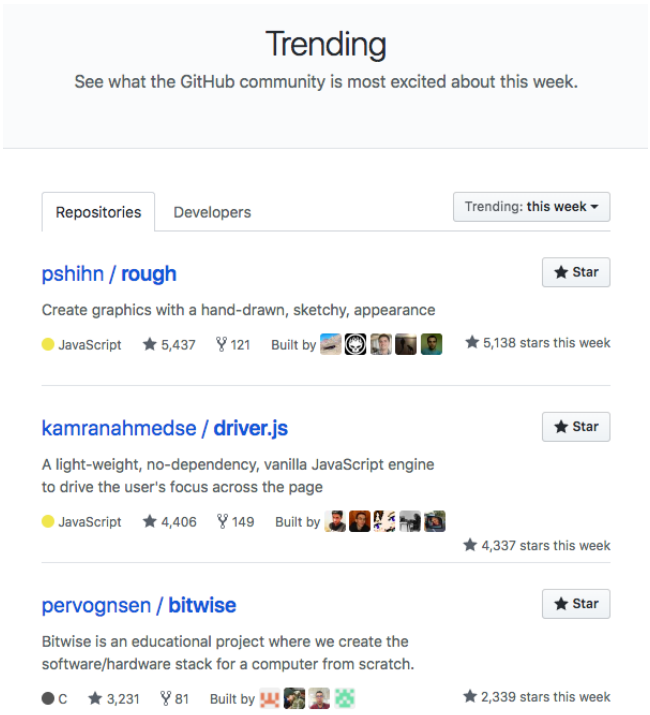


Figure 2.4: List of trending repositories on GitHub

Finally, GitHub provides a social feature named *stars*, which is similar to a *like* in other social media sites (e.g., Facebook, Twitter, and HackerNews). GitHub defines stars as a way to show appreciation to the repository's maintainers for their work and/or to make it easier to find a repository later. Indeed, stars play an important role

in the GitHub platform and most of the repository rankings depend on this measure.[2] Additionally, GitHub has a dedicated page where they list the repositories that gained more stars in the day, week, or month (as shown in Figure 2.4).

However, we still have few studies about the popularity of open source projects in social coding platforms. Weber and Luo [2014] attempt to differentiate popular and unpopular Python projects on GitHub using machine learning techniques. They found that in-code features are more important than author metadata features. Zhu et al. [2014] study the frequency of folders used by 140 thousands GitHub projects and their results suggest that the use of standard folders (e.g., doc, test, examples) may have an impact on project popularity. Bissyande et al. [2013] analyze the popularity, interoperability, and impact of various programming languages, using a dataset of 100K open source software projects. Aggarwal et al. [2014] study the effect of social interactions on GitHub projects' documentation. They conclude that popular projects tend to attract more documentation collaborators. Jiang et al. [2017] 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. Ma et al. [2016] conduct a study to identify the most influential Python projects on GitHub. They found that these projects are not necessarily popular among GitHub users. Papamichail et al. [2016] argue that the popularity of software components is as an indicator of software quality. As one of the findings of a systematic mapping study, Cosentino et al. [2017] report that popularity is also useful to attract new developers. Finally, by analyzing the effect of evolutionary software requirements on open source projects, Vlas et al. [2017] state that the success depends on the continuous developing of requirements.

A deep understanding of technical and non-technical factors that impact the popularity can provide valuable insights on how to build and evolve systems in a competitive market. In the next section, we discuss studies the popularity aspect in other platforms (e.g., app stores and social media sites) and studies that investigate the relationship between popularity and software quality.

2.2 Software Popularity

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 their code properties [Datta and Kajanan, 2013; Fu et al., 2013; Linares-Vasquez et al., 2013; Mojica Ruiz et al., 2014; Lee and Raghu, 2014; Tian et al., 2015; Guerrouj

[2]https://help.github.com/articles/about-stars, verified on 03/12/18.

et al., 2015; Palomba et al., 2015; Corral and Fronza, 2015; McIlroy et al., 2016]. Tian et al. [2015] investigate 28 factors along eight dimensions to understand how high-rated Android applications are different from low-rated ones. Their results show that external factors, like number of promotional images, are the most influential ones. Guerrouj and Baysal [2016] explore the relationships between mobile apps' success and API quality. They found that changes and bugs in API methods are not strong predictors of apps' popularity. In another study, Guerrouj et al. [2015] analyse changes of Android API elements between releases and report that high app churn leads to lower user ratings. McIlroy et al. [2016] study the frequency of updates in popular free apps from different categories in the Google Play store. They report that frequently-updated apps do not experience an increase in negative ratings by their users. Mojica Ruiz et al. [2014] examine the relationship between the number of ad libraries and app's user ratings. Their results show that there is no relationship between the number of ad libraries in an app and its rating. Linares-Vasquez et al. [2013] investigate how the fault- and change-proneness of Android API elements relate to applications' lack of success. They state that making heavy use of fault- and change-prone APIs can negatively impact the success of these apps. Lee and Raghu [2014] tracked popular apps in the Apple Store and found that the survival rates of free apps are up to two times more than the paid ones. Moreover, they report that frequent feature updates can contribute to app survival among the top ones. Ali et al. [2017] conducted a comparative study of cross-platform apps to understand their characteristics. They show that users can perceive and rate differently the same app on different platforms.

Other studies track popularity on social networks, including video sharing sites (e.g., YouTube) and social platforms (e.g., Twitter). Chatzopoulou et al. [2010] analyze popularity of YouTube videos by looking at properties and patterns metrics. They report that several popularity metrics are highly correlated. Figueiredo et al. [2011] characterize the growth patterns of videos on YouTube. They show that copyright protected videos tend to get most of their views earlier and videos on top lists tend receive a large fraction of their views on a peak day or week. Lehmann et al. [2012] analyze popularity peaks of hashtags in Twitter. For example, they found four usage patterns restricted to a two-week period centered on the peak times. Babaei et al. [2018] study news posts that reach a consensus (i.e., evoke similar reactions) from readers on Twitter. By analyzing the popularity of news posts with high and low consensus, they show that both types of news are equally popular, in number of retweets, with users.

Finally, other studies analyze the relationship between popularity and software quality. Sajnani et al. [2014] study the relationship between component popularity and component quality in Maven, finding that, in most cases, there is no correlation. Capra

et al. [2011] evaluate the effect of companies participation on open source communities and conclude that this involvement improves the popularity, but leads to lower software quality.

2.3 Popularity Prediction

One of the chapter of this thesis (Chapter 5) is inspired by the vast literature on defect prediction. For example, a systematic literature review listed 208 defect prediction studies [Hall et al., 2012], which differ regarding the software metrics used for prediction, the modeling technique, the granularity of the independent variable, and the validation technique. As independent variables, the studies use source code metrics (size, cohesion, coupling, etc), change metrics, process metrics, code smells instances, etc. The modeling techniques vary with respect to linear regression, logistic regression, naive bayes, neural networks, etc. In this thesis, instead of predicting the future number of defects of a system, we rely on multiple linear regressions to predict the number of stars of GitHub repositories.

Jiang et al. [2016] explore *why* and *how* developers fork *what* from *whom* in GitHub. They report that some repository owners are popular, and attract many forks; other owners are unpopular and rarely attract forks. They also report that a higher percentage of attractive owners are organizations, have more followers and earlier registration in GitHub.

Martin et al. [2016] extract time series information about popular Google Play apps and investigate how release frequency affects an app's performance, as measured by rating, popularity, and number of user reviews. They label as "impactful releases" the ones that caused a significant change on the app's popularity, as inferred by Causal Impact Analysis (a form of causal inference). They report that more mentions of features and fewer mentions of bug fixing increase the chance for a release to be impactful. Couto et al. [2014] follow a similar approach but to identify causal relationships between changes in internal measures of software quality (coupling, cohesion, complexity, etc) and the number of defects reported for a system.

Popularity prediction in other social networks is the target of several studies. In Twitter, Ma et al. [2013] predict hashtag popularity to identify fast emerging topics attracting collective attention. Their results reveal that context features (e.g., number of users that tweeted the hashtag) are relatively more effective than content feature (e.g., number of tweets with the hashtag). Tsur and Rappoport [2012] used a hybrid approach based on linear regressions to predict the spread of ideas in Twitter and found

that a combination of content features with temporal and topological features minimizes prediction error. In YouTube, Szabo and Huberman [2010] found a strong linear correlation between the popularity of videos at early and later times. Based on this finding, they present a model to predict future popularity. Pinto et al. [2013] propose two prediction models based on multivariate linear regression that incorporate information about historical patterns. Finally, Roy et al. [2013] propose a framework called *SocialTransfer* that utilizes knowledge from social streams (e.g., Twitter) to discover sudden popularity bursts in videos. They show that social trends have a ripple effect as they spread from the Twitter domain to the video domain. To our knowledge, this thesis is the first to target popularity prediction—measured by the number of stars—of software projects in the GitHub social coding network.

2.4 Software Promotion

Although open source software has been exhaustively explored recently, little is known about how developers promote these projects. Bianco et al. [2012] analyze marketing and communication strategies of three companies that develop open source software. By means of interviews, they found that websites and product launch events are adopted by the three organizations; however, the organizations differ considerably on the use of other communication channels, mainly when promoting the projects in open source communities and among industrial users. Singer et al. [2014] report a qualitative study focused on discovering the benefits that Twitter brings to developers. They found that Twitter adopters use it to stay aware of industry changes, for learning, and for building relationships. By correlating the blogging and committing behavior of developers, Pagano and Maalej [2011] observed an intensive use of blogs, frequently detailing activities described shortly before in commit messages. Bajic and Lyons [2011] analyze how software companies use social media techniques to gather feedback from users collectively. Their results suggest that startups use social media mainly for competitive advantage and established organizations use it to monitor the buzz among their users. By studying a successful software development company, Hansson et al. [2006] identified that user meetings and newsletters are adopted to include and increase the participation of users in the development process. Finally, Aniche et al. [2018] conduct a study to understand how developers use modern news aggregator sites (Reddit and Hacker News). According to their results, the two main reasons for posting links on these sites is to promote own work and to share relevant content.

2.5 Concluding Remarks

In this chapter, we provided background information and related work to better understand the state of the art related to this PhD thesis. Firstly, we introduced the concept of social coding and how this concept is influencing modern software development (Section 2.1). Moreover, we presented some of the social features available on GitHub, the most popular social coding platform, and studies that investigate the impact of these features on software development. In Section 2.2, we showed that popularity has been explored in several other contexts (e.g., app stores and social media sites) and with different purposes (e.g., correlate popularity and internal software quality). In Section 2.3, we presented existing investigations on defect prediction, which was one of the inspiration for a part of this thesis, in different contexts (e.g., social networks and app stores). Finally, we discussed studies on software promotion, since we also investigate the most common channels used by open source developers to promote their projects (Section 2.4).

Chapter 3

Characterizing the Popularity of GitHub Projects

In this chapter, we report a deep investigation on the popularity of GitHub repositories. In Section 3.1, we introduce the study. Next, Section 3.2 describes and characterizes the dataset used in the study. Then, in Section 3.3 we report a survey with developers to reveal their motivations for starring GitHub projects and in Section 3.4 we describe a quantitative characterization of the number of stars of GitHub projects. Furthermore, in Section 3.5 we propose a set of patterns to describe the popularity growth of GitHub repositories. In Section 3.6, we identify factors that distinguish the repositories in each growth pattern and in Section 3.7 we report a survey with developers to reveal their perceptions on these patterns. Finally, in Section 3.8 we discuss threats to validity and Section 3.9 concludes the chapter.

3.1 Introduction

Software popularity is a valuable information to modern open source developers, who constantly want to know if their systems are attracting new users, if new releases are gaining acceptance, or if they are meeting user's expectations. In order to provide initial evidence on the most useful metrics for measuring the popularity of GitHub projects, we conducted an initial survey with StackOverflow users. We rely on these users because StackOverflow is a widely popular programming forum, listing questions and answers about a wide variety of technologies, which are provided by practitioners with different profiles and background [Vasilescu et al., 2013]. We randomly selected a sample of 400 StackOverflow users, using a dump of the site available at Google

BigQuery.[1] We e-mailed these users asking them 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. Although available in any repository, project owners do not have control about 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 3.1, the results of this initial survey 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).

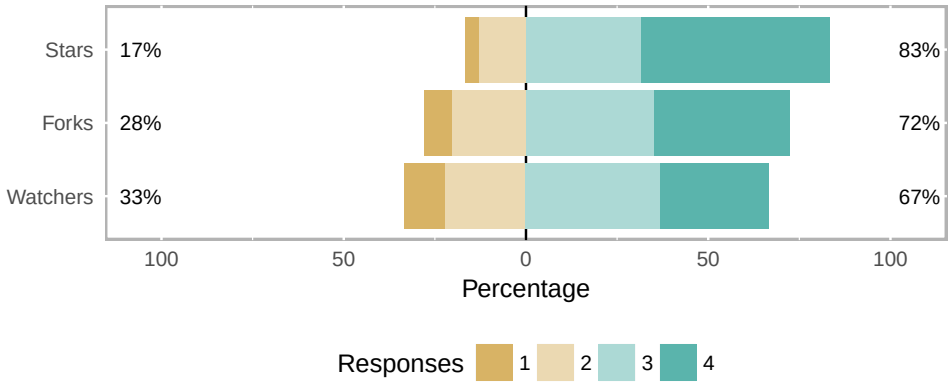


Figure 3.1: How useful are the following metrics to assess the popularity of GitHub projects? (1: not useful; 4: very useful)

After confirming the relevance of stars in the preliminary survey, we conducted a large-scale characterization study, with GitHub projects and developers, which is described in details in this chapter. In the first part of this characterization study, we collected historical data about the number of stars of 5,000 popular GitHub repositories. We perform a survey with 791 developers to reveal their motivations for starring projects. Our intention is to confirm that stars are a reliable measure of popularity on GitHub. Moreover, we use our dataset to answer five research questions:

RQ #1: How popularity varies per programming language, application domain, and repository owner? The goal is to provide an initial view about the popularity of

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

the studied systems, by comparing the number of stars according to programming language, application domain, and repository owner (user or organization).

RQ #2: Does popularity correlate with other characteristics of a repository, like age, number of commits, number of contributors, and number of forks? This investigation is important to check whether there are factors that can be worked to increase a project's popularity.

RQ #3: How often do repositories lose popularity? On GitHub, users can remove their stars from previously starred repositories (which is commonly called an unstar event). The goal of this third research question is to explore how often developers unstar repositories and correlate this information with the programming language and application domain.

RQ #4: How early do repositories gain popularity? With this research question, we intend to check whether gains of popularity are concentrated in specific phases of a repository's lifetime, specifically in early releases.

RQ #5: What is the impact of new features on popularity? This investigation can show if relevant gains in popularity happen due to new features, as implemented in new releases.

In the second part of the study, we propose four patterns of popularity growth in GitHub, which are derived after clustering the time series that describe the growth of the number of stars of the projects in the dataset. We also investigate the endogenous factors (i.e., the ones that can be retrieved directly from a repository) that affect the classification of a project in a given growth pattern. Finally, in the third part of the study, we present a survey conducted with 115 project owners to reveal their perceptions about the proposed growth patterns. Our intention is to derive actionable insights to developers interested in increasing the number of stars of their projects.

3.2 Dataset

The dataset used in this study 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 highly popular GitHub projects. Second, because we investigate the impact of application domain on popularity, 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 3.5, 3.6, and 3.7.

Table 3.1 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.

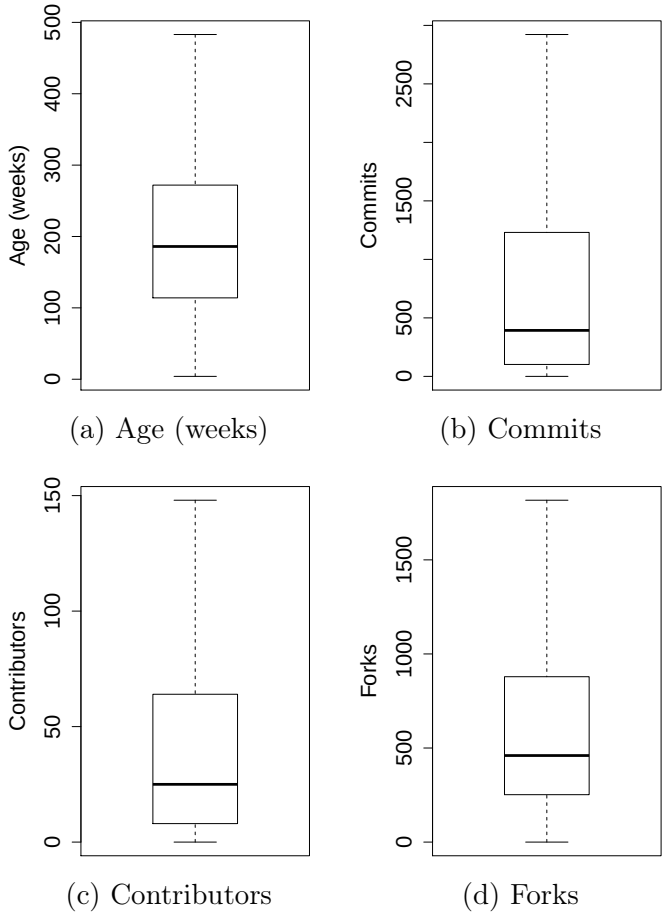
Table 3.1: Descriptive statistics on the number of stars of the repositories in our dataset

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

Age, Commits, Contributors, and Forks: Figure 3.2 shows boxplots about 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;[2] 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 3.3 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

[2]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.



(a) Age (weeks) (b) Commits

(c) Contributors (d) Forks

Figure 3.2: Age, number of commits, number of contributors, and number of forks (outliers are omitted)

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).

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 author of this thesis 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

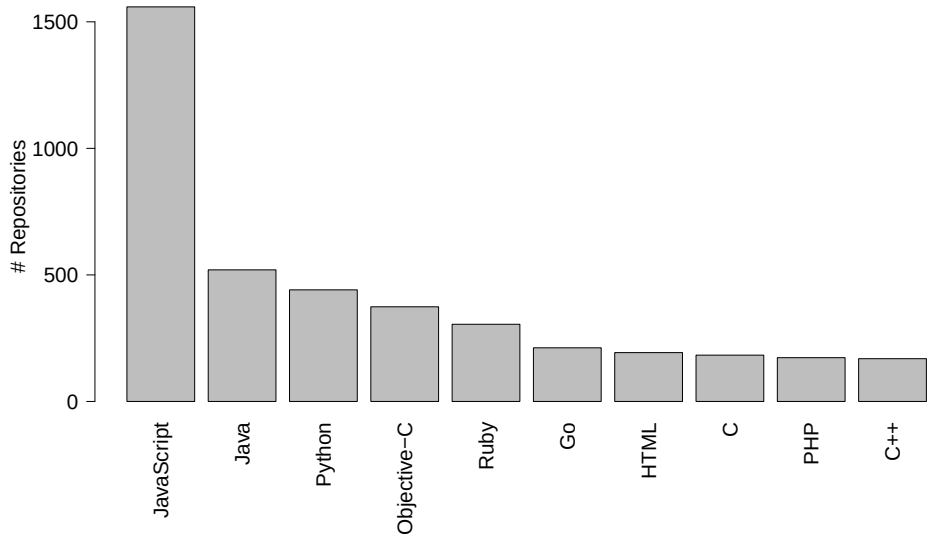


Figure 3.3: Top-10 languages by number of repositories

domains were validate with a second researcher. 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:[3]

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).

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

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 3.4 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.

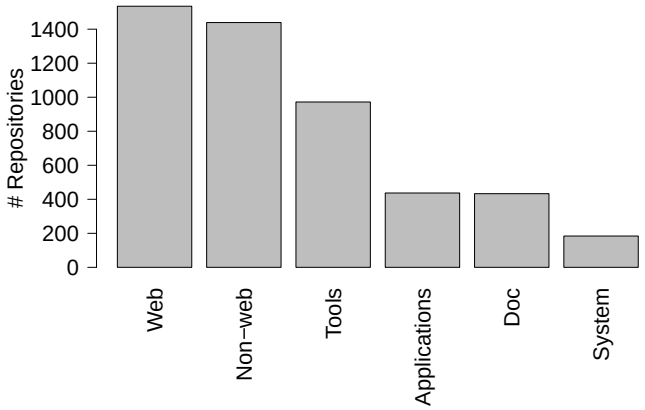


Figure 3.4: Number of repositories by domain

3.3 Survey Study

We conducted 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. In this section, we describe the design of the survey questionnaire and the selection of the survey participants (Section 3.3.1) and report the survey results (Section 3.3.2).

3.3.1 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 investigate whether stars can be viewed as a measure of popularity, which we define as follows: "the state of being liked, enjoyed, accepted, or done by a large number of people", according to Merriam-Webster. 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 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.

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. To preserve the respondents privacy, we use labels P1 to P791 when quoting the answers. We analyze the answers using thematic analysis [Cruzes and Dyba, 2011], 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 thesis.

3.3.2 Survey Results

A separate subsection discusses the answers to each survey question.

3.3.2.1 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 3.2 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.

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%).

Table 3.2: 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

3.3.2.2 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. 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, 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 3.3, 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 3.3: 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., popularity does not reflect the 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. *Practical Implication*: Stars are a reliable proxy for the popularity of GitHub projects.

3.4 Characterization Study

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

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

RQ #2: Does popularity correlate with repository's age, number of commits, number of contributors, and number of forks? This investigation is important to check whether there are factors that can be worked to increase a project's popularity.

RQ #3: How often do repositories lose popularity? On GitHub, users can remove their stars from previously starred repositories (which is commonly called an unstar event). The goal of this third research question is to explore how often developers

unstar repositories and correlate this information with the programming language and application domain.

RQ #4: How early do repositories gain popularity? With this research question, we intend to check whether gains of popularity are concentrated in specific phases of a repository's lifetime, specifically in early releases.

RQ #5: What is the impact of new features on popularity? This investigation can show if relevant gains in popularity happen due to new features, as implemented in new releases.

The proposed research questions aim to shed light on the relation between GitHub stars and other project metrics and characteristics. Some our findings directly support actionable guidelines, e.g., we reveal that repositories owned by organizational accounts are more popular. Others motivate or support developers when making a decision to start a new open source project, e.g., we show that there is still a relevant demand for Web libraries and frameworks, mostly implemented in JavaScript. Furthermore, for existing repositories, our findings can explain their popularity, but cannot be used to improve it; for example, it is not practical to migrate a project to a new programming language and, most obviously, to a new application domain.

3.4.1 Results

RQ #1: How popularity varies per programming language, application domain, and repository owner?

Figure 3.5 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 3.5, 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*).

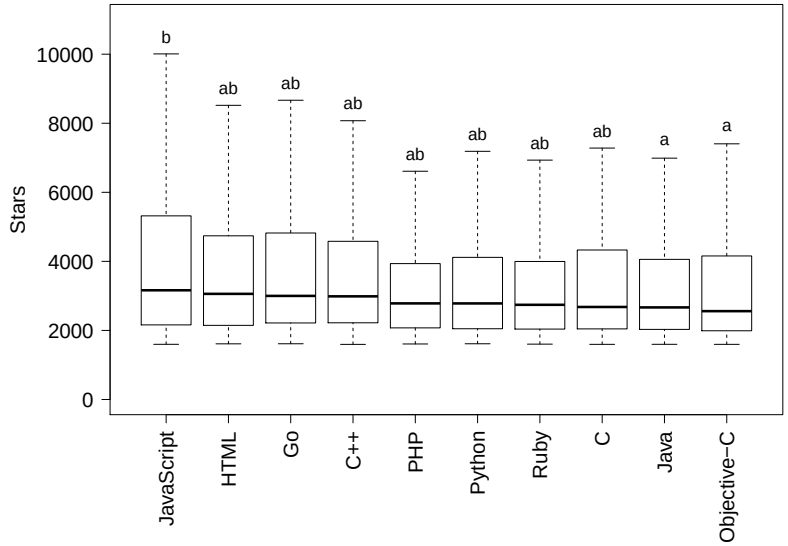


Figure 3.5: Stars by programming language (considering only the top-10 languages with more repositories)

Figure 3.6 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 non-web libraries and frameworks (2,642 stars). By applying the Kruskal-Wallis test, we found that the distributions are different ($p\text{-}value < 0.001$). 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 from this domain are less popular. Similarly, tools (label *b*) are more popular only than non-web libraries and frameworks (label *c*). Finally, there is no statistical difference between the popularity of systems software, applications, web libraries and frameworks, and documentation (since all these distributions have the label *a* in common).

Finally, Figure 3.7 shows how popularity 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 ($p\text{-}value < 0.001$) with a *very small* effect size (Cohen's $d = -0.178$). We hypothesize that repositories owned by organizations—specifically major software companies and free software foundations—have more funding and resources, which contributes to their

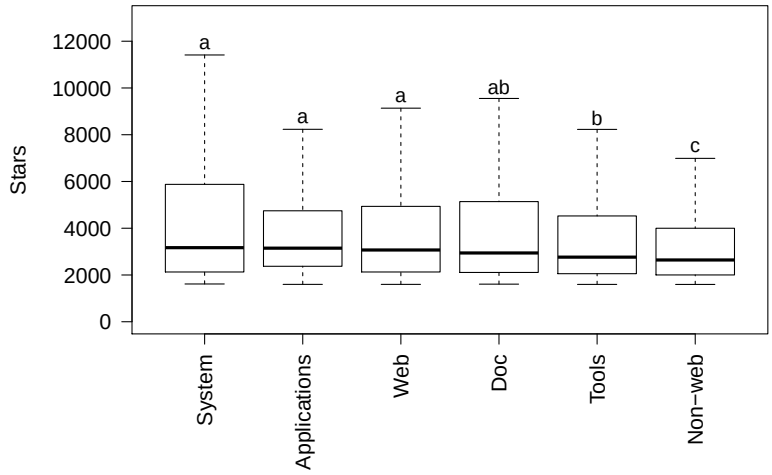


Figure 3.6: Popularity by application domain

higher popularity.

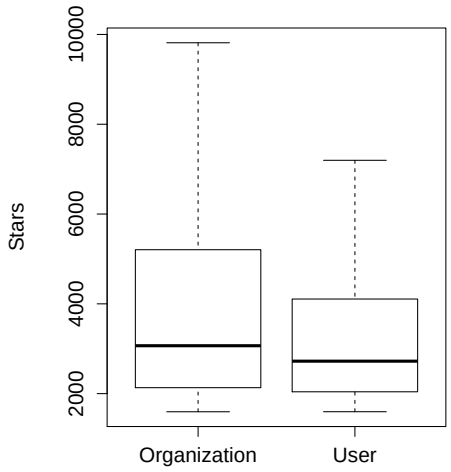


Figure 3.7: Popularity by repository owner

Among the top-100 most popular repositories, only 29 repositories are owned by users. The developers of 17 of such systems have a public email address in their GitHub profile. We sent a short questionary to these developers and received responses from five of them (29.8%). In this questionary, we asked two questions. First, we asked the developers about possible plans to migrate their repositories to an organization account. All developers answered negatively this question. Two developers mentioned they want to explicitly appear as the repository owner, like in this answer:

"I worked hard to create the project, and having it under my personal username is necessary to have proper credit for it."

To complement the first question, we asked the developers if they agree that migrating the repositories to an organization account would help to attract more users. Four developers (80%) answered negatively to this question and only one participant provided the following answer:

"It depends on what organization it is. If it's a well known org I'm sure it helps, otherwise I don't think it makes a difference."

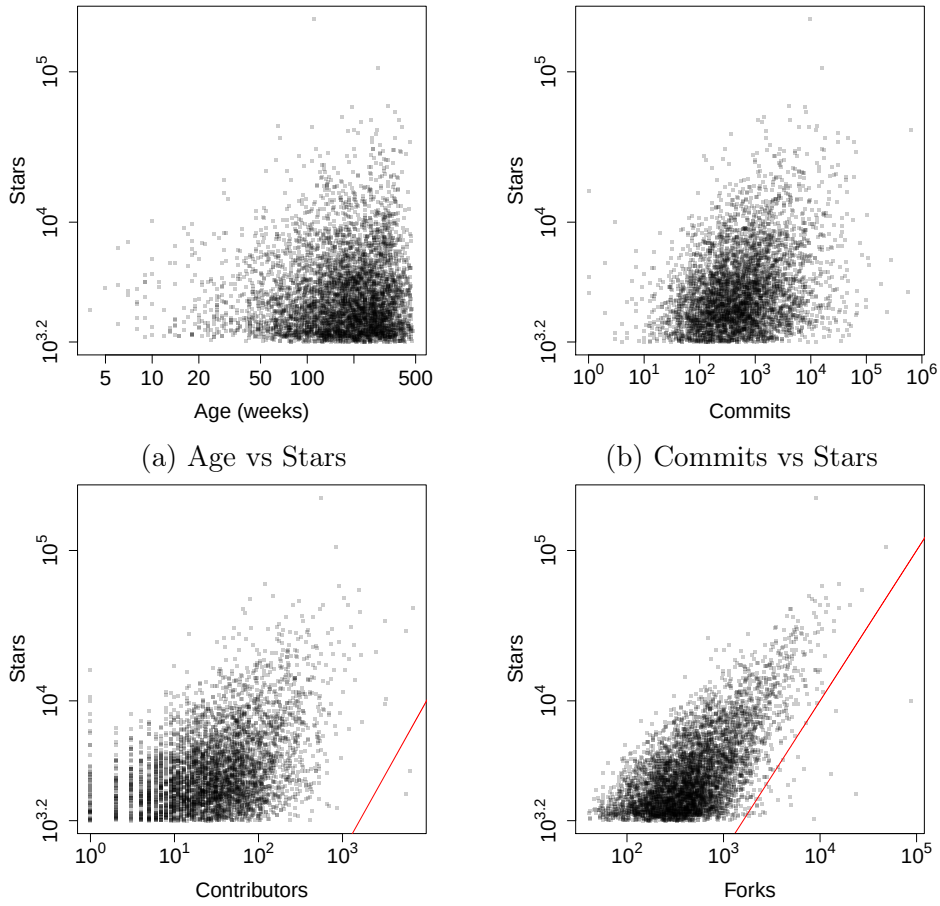
Therefore, although it seems "easier" to organizations to reach the top positions of GitHub popularity ranking, some projects owned by individual developers also reach these positions. These developers usually do not want to move to organizational accounts, basically to keep full control and credit for their repositories.

Summary: JavaScript is the language with the highest number of popular repositories (median of 3,163 stars). The top-3 most popular application domains are (1) systems software, (2) applications, and (3) web libraries and frameworks. Repositories owned by organizations are more popular than the ones owned by individuals. *Practical Implications:* Programming language and application domain may influence the popularity of a repository. On the one hand, projects developed in JavaScript are likely to have more stars than projects developed in other languages. On the other hand, non-web libraries and frameworks are expected to have less stars than projects in other application domains.

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

Figure 3.8 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. [2003], 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 3.8a). 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 3.8: Correlation analysis. In subfigures (c) and (d), the line is the identity relation

The scatterplot in Figure 3.8b 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 3.8c, 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 3.8d shows plots correlating a system popularity and its number of forks. As suggested by the followed guidelines, there is a moderate positive correlation between stars and forks ($rho = 0.558$ 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 3.8d, 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 the number of stars and the repository's age; however, there is a low correlation with commits, and a moderate correlation with contributors and forks. *Practical Implications:* Increasing the number of GitHub stars may help to attract more contributors to a project and increase its chances of long-term success.

RQ #3: How often do repositories lose popularity?

In this research question, we investigate how often developers remove stars from GitHub repositories. Moreover, we provide an estimation of the time taken by developers to lose interest on the projects (i.e., the time between the "star" and the "unstar" events). Finally, we correlate this information with programming language and application domain.

As GitHub does not provide information on unstar events, we implemented a script to identify users that removed their star by comparing the list of users starring the repositories in different dates. Basically, our goal was to identify users that removed their star and consequently are not present in the most recent list of users starring a given repository. In this investigation, we first collected the users starring the studied repositories on April 1, 2018, and computed the difference with the ones starring the repositories on April 30, 2018. Due to the interval between the dataset construction

and the data collection for the investigation, some of the repositories were not anymore available (e.g., they were moved, deleted or became private). On total, we analyzed 4,709 repositories, which represents 94.18% of the repositories in our dataset. In this period, these repositories gained a total of 489,338 stars and 54,025 unstars.

Figure 3.9 shows a scatterplot correlating the number of stars and unstars events per repository. As expected, most of the repositories gain more stars than lose stars (i.e., they are represented above the identity line in the plot). Specifically, 188 repositories (3.99%) lost more stars on the last month than gained stars in the same period. From this total, 173 repositories (92%) lost at most 10 stars, which suggest that unstar events do not have a major influence on repositories popularity. The repository that lost more stars was expressjs/express (1,354 stars and 730 unstars), followed by flutter/flutter (2,587 stars and 452 unstars), atom/atom (693 stars and 167 unstars), bitcoin/bitcoin (1,146 stars and 160 unstars), and Microsoft/vscode (2,032 stars and 159 unstars). By computing the Spearman's rank coefficient, we found a high correlation between stars gained and unstars ($rho = 0.730$ and $p\text{-}value < 0.001$). Thus, the number of unstars tend to be proportional to the number of stars gained.

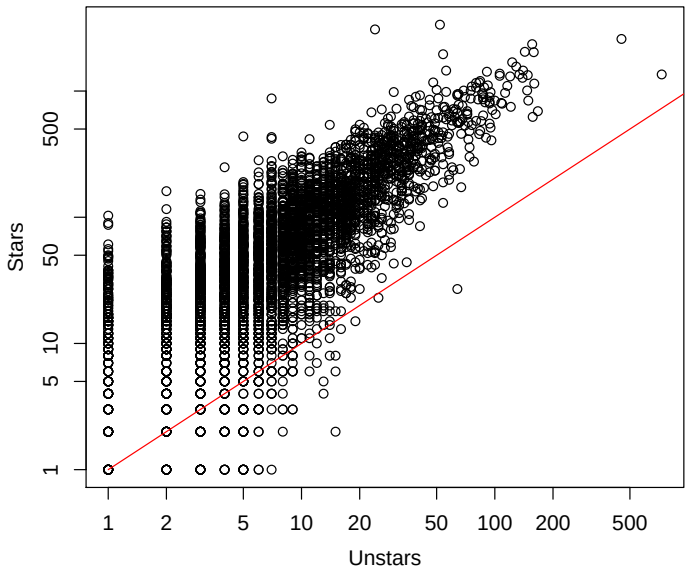


Figure 3.9: Number of stars and unstars events per repository (during April, 2018). The red line is the identity function.

We also estimate the time taken by developers to remove their stars from the repositories. We found that some users remove their stars few hours after starring the projects and other ones take decades to remove. For example, we found that a users took only 7 hours to remove its star from rg3/youtube-dl. By contrast, another removed its star from mislav/will_paginate after 10 years! On average, users take

1.13 years and, on median, 1.59 years to remove stars from projects. Figure 3.10 shows the distribution of the time, in years, that developers took to unstar repositories according to the programming language (restricted to the ones with more repositories in our dataset). The language who developers take more time, on median, to unstar repositories is Objective-C (2.27 years), followed by Ruby (2.02 years) and Java (1.17 years). By applying the Kruskal-Wallis test, we found that the distributions are different ($p\text{-}value < 0.001$). According to Dunn's test, the distribution of Objective-C (label f) and Ruby (label g) are statistically different from all other languages, suggesting that projects from these languages retain the developers interest for more time. By contrast, projects in C++ and Go presented the lowest values, with 295 days on median between stars and unstars events.

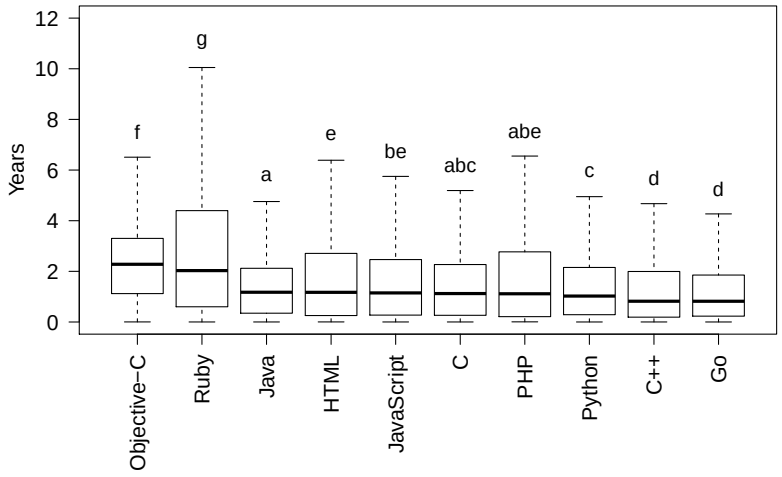


Figure 3.10: Years between stars and unstars events by programming language. Outliers are omitted.

Finally, Figure 3.11 shows the distribution of the time took by developers to unstar repositories, according to the application domain. The median time varies as follow: applications (1.17 years), web libraries and frameworks (1.17 years), non-web libraries and frameworks (1.16 years), software tools (1.11 years), systems software (1.03 years), and documentation (1.02 years). By applying the Kruskal-Wallis test, we found that the distributions are different ($p\text{-}value < 0.001$). According to Dunn's test, documentation and systems software (labels d and b, respectively) are statistically different from the other domains. Moreover, web and non-web libraries and frameworks are also statistically different (labels a and c, respectively).

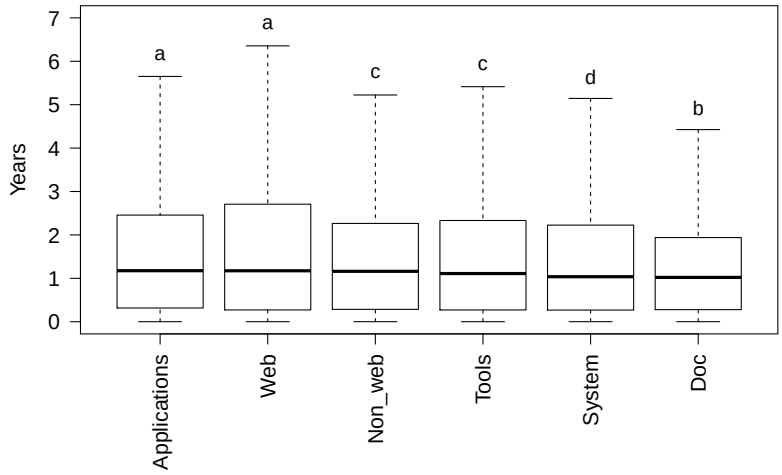


Figure 3.11: Years between stars and unstars events by domain. Outliers are omitted.

Summary: Unstar events do not have a major influence on the popularity of GitHub repositories. Objective-C is the language whose developers take more time to remove their stars. Moreover, projects from the documentation domain are those who developers lose their interest faster.

RQ #4: How early do repositories gain popularity?

Figure 3.12 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 3.12). 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 released. After this initial burst, the popularity growth of the repositories 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 total number of stars (label C).

Figure 3.13 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 quartile), and 2.7% (third quartile). By applying the Mann-Whitney test, we found that these distributions are different

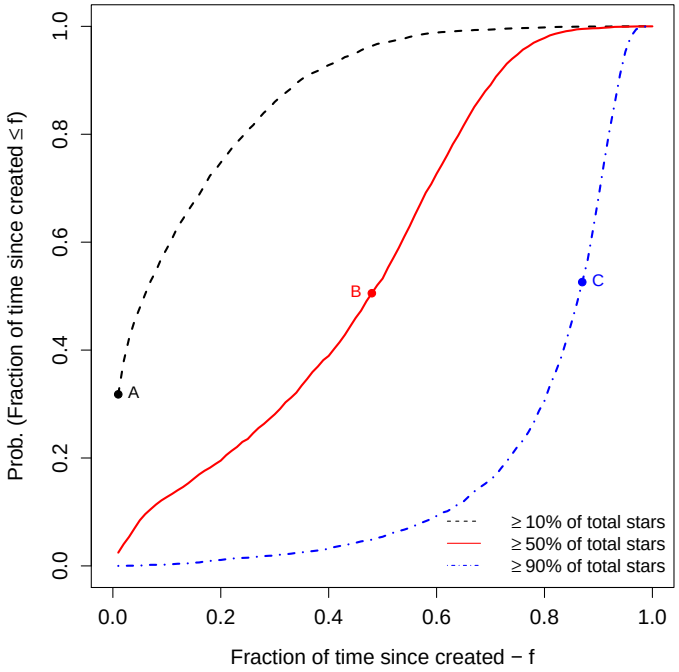


Figure 3.12: Cumulative distribution of the time fraction a repository takes to receive 10%, 50%, and 90% of its stars

(p-*value* < 0.001) with a *large* effect size (Cohen's $d = 0.856$).

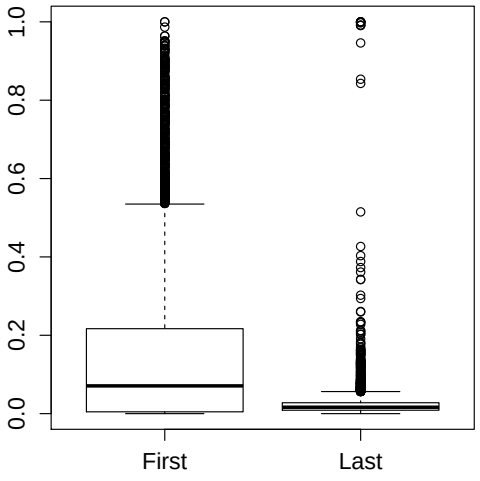


Figure 3.13: Fraction of stars gained in the first four weeks and in the last four weeks

Summary: Repositories have a tendency to receive more stars right after their first public release. After this period, the growth rate tends to stabilize. *Practical Implications:* GitHub developers should concentrate more efforts to promote their projects in the first weeks after the initial release.

RQ #5: What is the impact of new features on popularity?

In this research question, we investigate the impact of new features on the popularity 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 popularity. 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 3.14, 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 3.15 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).

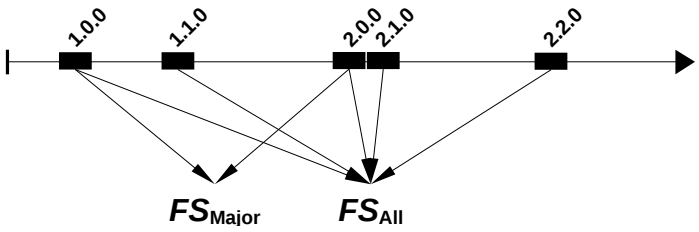


Figure 3.14: Fraction of stars for all releases (FS_{All}) and just after major releases (FS_{Major})

Figure 3.16 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

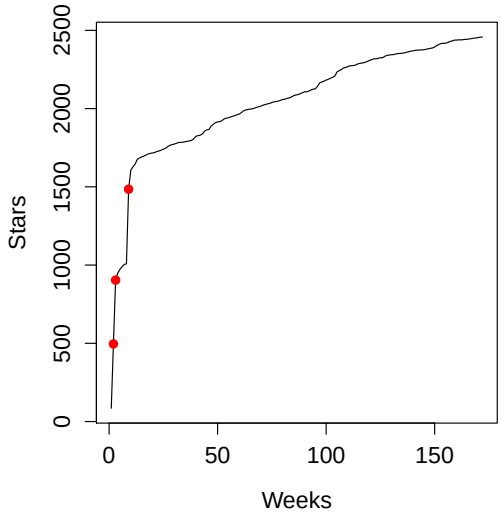


Figure 3.15: REPORTR/DASHBOARD (the dots indicate weeks with releases)

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.

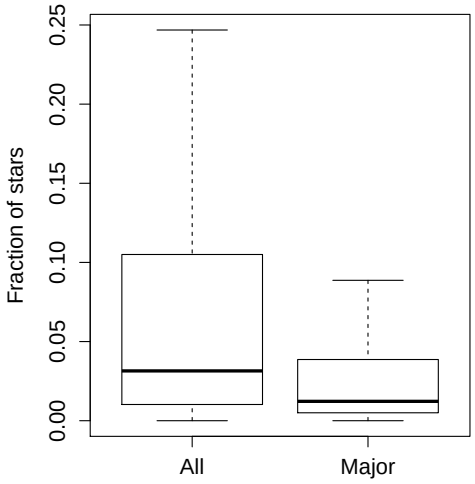


Figure 3.16: Fraction of stars gained in the first week after all releases and just after the major releases

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 3.15) 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 3.17 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 ($d = -0.188$).

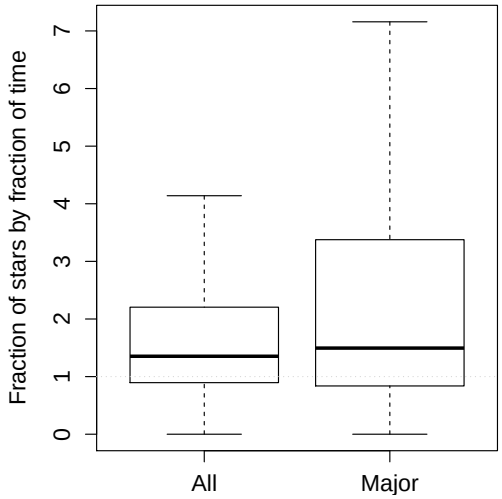


Figure 3.17: Fraction of stars gained in the week following all releases (or just the major releases) / fraction of time represented by these weeks

Figure 3.18 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 are some gains of stars after releases, they tend to decrease after few weeks.

Finally, to reveal the characteristics of the most successful releases, we sent a brief questionary to the main developers of 60 releases with the highest fraction of stars gained on the week after the release (and whose developers have a public email address on their GitHub profile). We received answers from 25 developers. First, we asked the developers about the type of features implemented in these releases. As presented in Table 3.4, the releases usually include both functional and non-functional requirements (14 answers), followed by releases with mostly functional requirements (9 answers). We did not receive answers about releases including non-functional re-

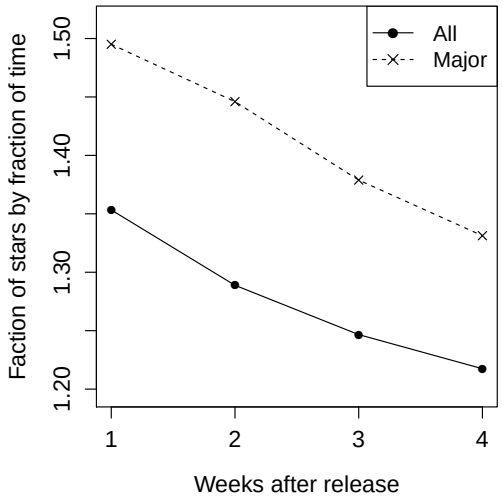


Figure 3.18: Fraction of stars by fraction of time (median values), computed using different time intervals

quirements. Two developers provide other types of answers ("complete rewrite" and "maintenance release", respectively).

Table 3.4: Features implemented in successful releases

Features	Answers
Both functional and non-functional	14
Mostly functional	9
Other answers	2
Mostly non-functional	0

We also asked the developers to explain how the features implemented in these releases were selected (answers including multiple items are possible in this question). As presented in Table 3.5, the features usually come from ideas of the repository' maintainers (23 answers) and from user's suggestions (11 answers).

Table 3.5: How the features are selected?

Features selected from	Answers
Ideas of the repository maintainers	23
Users suggestions	11
Features of similar projects	6
Other answers	3

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 the other weeks. However, because repositories usually have more weeks without releases, this phenomenon is not sufficient to generate a major concentration of popularity gain after releases. For example, 75% of the systems gain at most 3.8% of their stars in the week following major releases. *Practical Implications:* Software releases can also be strategically used to increase the repository' popularity.

3.5 Popularity Growth Patterns

In this section, we investigate common popularity growth patterns concerning the GitHub repositories in our dataset. To this purpose, we use the KSC algorithm [Yang and Leskovec, 2011]. 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 [Figueiredo, 2013] and Twitter posts [Lehmann et al., 2012]. Like K-means [Hartigan, 1975], 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 in 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 3.7. 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 Menasce and Almeida [2001] 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 [Figueiredo et al., 2014]. In our dataset, the values of β_{CV} start to stabilize for $k = 4$ (see Fig-

ure 3.19). 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.

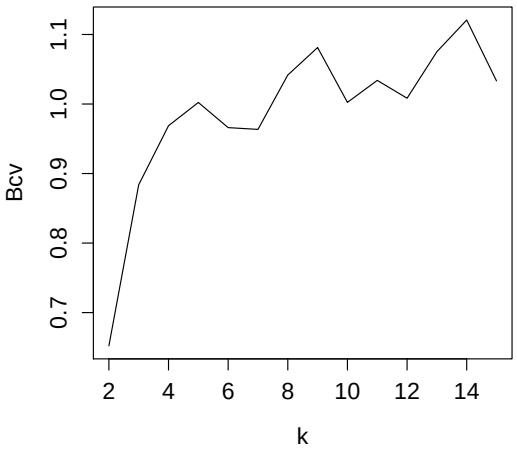
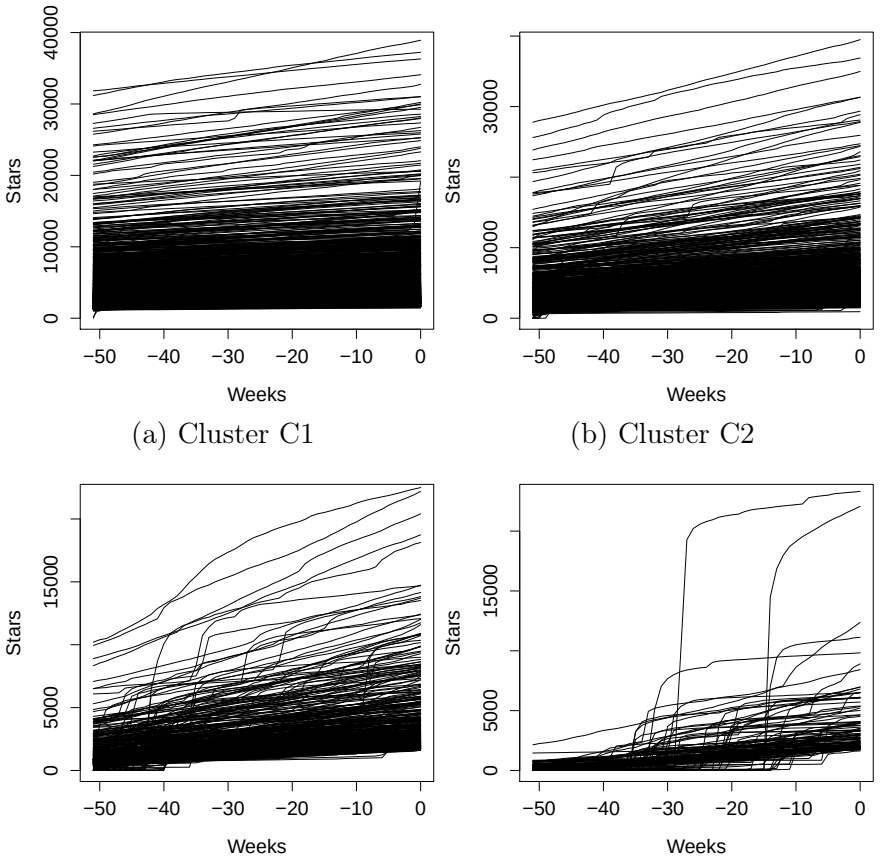


Figure 3.19: β_{CV} for $2 \leq k \leq 15$

3.5.1 Proposed Growth Patterns

Figure 3.20 shows plots with the time series in each cluster. The time series representing the clusters' centroids are presented in Figure 3.21. 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 on 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 3.6. 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 on 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 3.22 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



(a) Cluster C1 (b) Cluster C2

(c) Cluster C3 (d) Cluster C4

Figure 3.20: Clusters of time series produced by the KSC algorithm

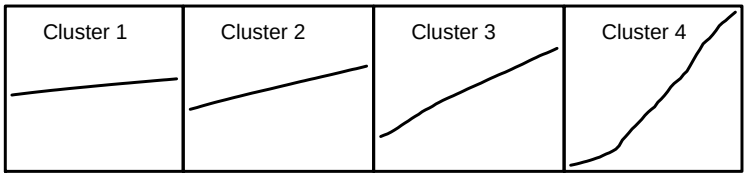
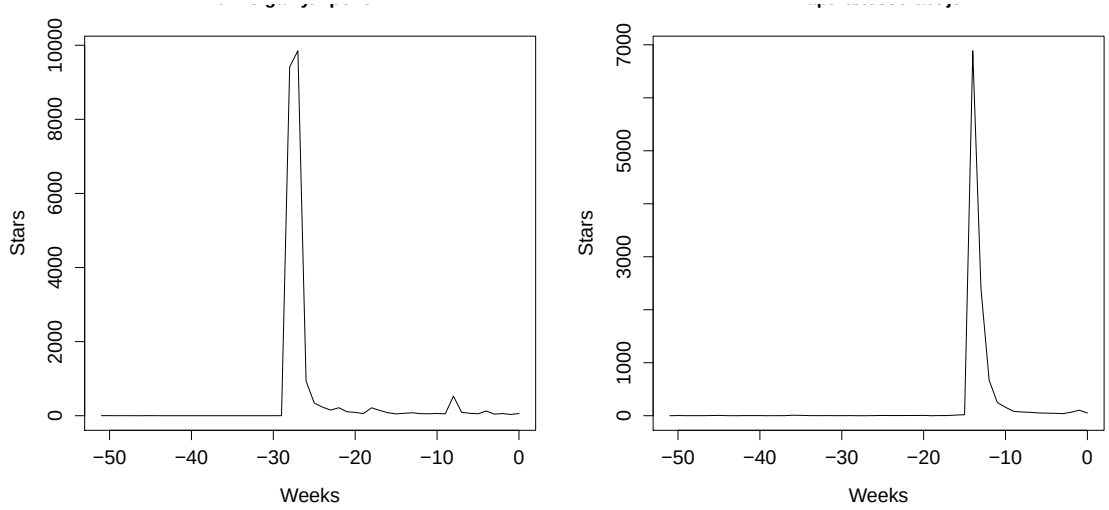


Figure 3.21: Time series representing the centroids of each cluster

naptha/tesseract.js (a JavaScript library to recognize words in images, which received 6,888 stars in a single week).

Table 3.6: Popularity 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



(a) APOLLO-11 (b) TESSERACT.JS

Figure 3.22: Examples of systems with viral growth

We also investigate the impact of the proposed growth patterns on repositories ranking. 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 3.23 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 repositories (109 out of 110) with viral growth improved their rank on 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.

3.6 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.

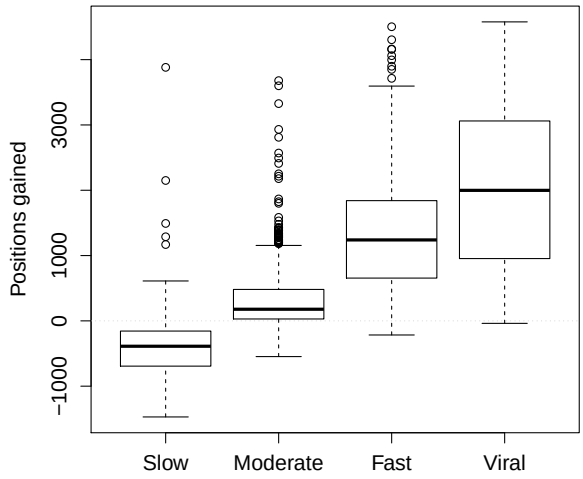


Figure 3.23: Rank differences in the interval of one year

3.6.1 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 [Breiman, 2001]. We choose Random Forest because it is robust to noise and outliers [Provost and Fawcett, 2001; Tian et al., 2015; Hora et al., 2016].

Table 3.7 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.

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[7], since their recent activities are shown to more developers [Tsay et al., 2014]. Finally, developers owning popular repositories 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

[4]Total number of forks including forks of forks

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

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

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

Table 3.7: 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[4]
	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[5] (boolean value)
	Is Mirror (r.mirror)	Repository is a mirror (boolean value)
	Domain (r.domain)	Application domain (as defined in Section 3.2)
	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[6]
	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
	Repositories Popularity (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

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 3.7. This technique is proposed for assessing features collinearity and it is used in several other studies [Tian et al., 2015; Rakha et al., 2016]. Figure 3.24 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).

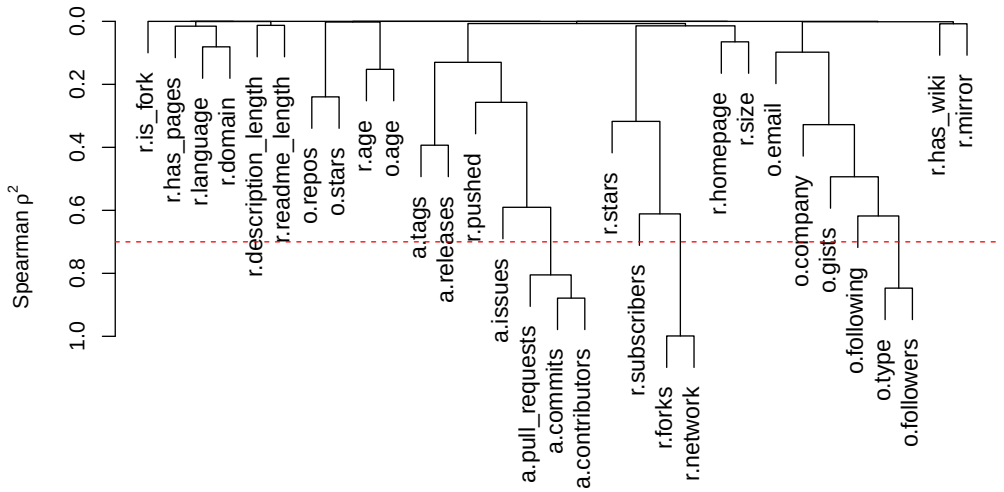


Figure 3.24: Correlation analysis (as result, we removed features *a.pull_requests*, *a.contributors*, *r.network*, and *o.type*)

3.6.2 Most Influential Factors

To assess the relative importance of the selected features in discriminating each growth pattern, we used the `rfPermute` package for `R` [Archer, 2013]. 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 [Wolpert and Macready, 1999].

Table 3.8 lists the top-10 most influential factors according to the feature importance ranking (all of them with $p\text{-}value < 0.01$). 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 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 3.8: Top-10 most influential factors (*p-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 3.8, 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 implies 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 updating and improving 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 [Coelho and Valente, 2017].

Finally, to assess the effectiveness of the classifier, we relied on metrics commonly used in Machine Learning and Information Retrieval [Yates et al., 1999]. 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 3.9 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 3.9: 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. *Practical Implications:* Active development is essential to popularity growth and competitiveness on GitHub.

3.7 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 3.7.1 describes the design of the survey questionnaire and the selection of the survey participants. Section 3.7.2 reports the survey results.

3.7.1 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 3.10). To preserve the respondents privacy, we use labels R1 to R115 when quoting their answers. After receiving the answers, the thesis's author analyzed them, following the same steps of the survey presented in Section 3.3.

Table 3.10: 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

3.7.2 Survey Results

Table 3.11 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)

After analyzing the reasons for *moderate* growth, we identified two conflicting sentiments in the answers: (a) *positives* reasons, which are contributing to the popularity

Table 3.11: 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

growth; (b) *negative* reasons, which are limiting the popularity growth. The major reasons for the *positive* and *negative* sentiments are listed in Tables 3.12 and 3.13, respectively.

For *positive* sentiments, 15 developers (45.4%) mentioned active promotion (mainly on social media sites, as Hacker News[8]). 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 popularity 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)

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)

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. Fi-

[8] `https://news.ycombinator.com`

Table 3.12: Reasons for Moderate Growth – Positive Sentiments – (95% confidence level with a 16.9% confidence interval)

Reason	Answers	Percentage (%)
Active promotion	15	45.4
Trending technology	9	27.2
Active project	7	21.2
Innovative project	3	9.0
Code or doc. quality	2	6.0
Other	3	9.0

Table 3.13: Reasons for Moderate Growth – Negative Sentiments – (95% confidence level with a 16.9% confidence interval)

Reason	Answers	Percentage (%)
Niche audience	3	9.0
Low activity	2	6.0
Limited or lack of promotion	1	3.0
Old project	1	3.0

nally, one developer mentioned that the project age is restricting its popularity 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 3.14 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)

Table 3.14: 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

Finally, Table 3.15 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)

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

Table 3.15: 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%). *Practical Implications:* Active promotion is an important aspect that should be emphasized by project managers. However, it is also important to provide high-quality features, code, and documentation.

3.8 Threats to Validity

Dataset. GitHub has millions of repositories. We built 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 to investigate the popularity of the most starred repositories. Furthermore, most GitHub repositories are forks and have very low activity [Kalliamvakou et al., 2014, 2015; Cosentino et al., 2017].

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 two researchers.

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 thesis's 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 [Menasce and Almeida, 2001]. 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 3.6, 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. [2012].

3.9 Conclusion

In this study, we reported that developers star repositories due to three major reasons (which frequently overlap): to show appreciation to the projects, to bookmark a project, and because they are using the project. Furthermore, three out of four developers declared they consider the number of stars before using or contributing to GitHub projects (Section 3.3).

> *Actionable Insight #1:* Stars are a reliable popularity measure; therefore, project managers should track and compare the number of stars of their projects with competitor ones.

In Section 3.4, we presented a quantitative characterization of the top-5,000 most starred repositories. We found that repositories owned by organizations are more popular 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). Moreover, we show that unstar events do not influence significantly on the popularity growth of the repositories (RQ #3). Furthermore, repositories have a tendency to receive more stars right after their first public release (RQ #4). Finally, there is an acceleration in the number of stars gained after releases (RQ #5).

Actionable Insight #2: Project managers should consider using organizational accounts (e.g., *aserg-ufmg* instead of *hsborges*). It is also important to work to attract new contributors and to evolve the projects by frequently providing new releases.

We validated the proposed popularity growth patterns (Sections 3.5 and 3.6) 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 the project deprecation or lack of activity. 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 technology, and innovative projects. Finally, the major reason for *viral* growth is also promotion on social media.

Actionable Insight #3: Open source projects require an investment on marketing and advertisement, mainly in social networks and programming forums, like Hacker News.

The analyzed data, manual classification of the application domain, and the surveyed responses used in this study is publicly available at the Zenodo data repository: `https://doi.org/10.5281/zenodo.1183752`.

Chapter 4

Promotion on Open Source Projects

In Chapter 3, we showed that active promotion is the major reason for moderate and fast growth in popularity, according to the surveyed developers. Therefore, in this chapter, we present an investigation on the most common channels used by developers to promote open source projects. This chapter is organized as follows. First, we motivate our study and present our research questions (Section 4.1). Then, we describe the methodology followed in the study (Section 4.2). Next, we present and discuss the results of each research questions (Section 4.3). Finally, Section 4.4 discusses threats to validity and Section 4.5 concludes the chapter.

4.1 Introduction

Open source projects have an increasing importance in modern software development. For example, several open source projects are daily used by millions of users. However, it is also important to continually attract participants and contributors to these projects, in order to increase the chances of long-term success [Comino et al., 2007]. Particularly, several channels can be used to promote open source software, helping to keep the interest of the community and also to attract new members. In this study, we intend to reveal the most common channels used by developers to promote open source projects. We address four research questions:

RQ #1: What are the most common promotion channels? The goal is to provide a list of the most common channels used to promote the open source projects on GitHub.

RQ #2: How often do developers promote their projects? With this research question, we intent to quantify the frequency that developers promote their projects on blogs and social networks. We also characterize the meeting groups of the projects' users.

RQ #3: How popular and random projects differ on the usage of promotion channels? This investigation can show if there is significant difference in the usage of the promotion channels by popular and non-popular GitHub projects.

RQ #4: What is the impact of promotion on Hacker News? Social news aggregators (e.g., Hacker News) are sites that allow developers to make their content easily reachable by a large audience. The goal of this final research question is to analyze the impact of successful posts on the number of stars of the analyzed projects.

4.2 Study Design

To reveal the most common promotion channels used by developers, we manually inspected the documentation of the top-100 projects with most stars on GitHub. We restricted our analysis to popular projects because they have a large number of users and therefore need better and efficient ways to communicate with users and also to attract new contributors.

Figure 4.1 shows the distribution of the number of stars of the projects considered in this study. This number ranges from 291,138 stars (FREECODECAMP/FREECODECAMP) to 23,322 stars (TIIMGREEN/GITHUB-CHEAT-SHEET). The considered projects are primarily developed on 17 programming languages; JavaScript is the most common one (40 projects), followed by Python (9 projects) and Go (5 projects). Furthermore, 14 projects only include markdown files with documentation purposes (e.g., projects with tutorials, books, awesome lists, etc). Finally, regarding the project owners, 69 are organizational accounts and 31 are user accounts.

For each of these 100 projects, we initially inspected their READMEs on GitHub to identify the channels used to promote the projects and to keep the users up-to-date with important information about them. For example, the following sentence is available on the README of ADOBE/BRACKETS: "*Y*ou can see some screenshots of Brackets on the wiki, intro videos on YouTube, and news on the Brackets blog". In this case, wiki and YouTube are used to support users whereas blog is a channel used to disseminate news about BRACKETS. Thus, only blog is considered a promotion channel in our study. Next, we inspected the projects' website, for those projects having one.

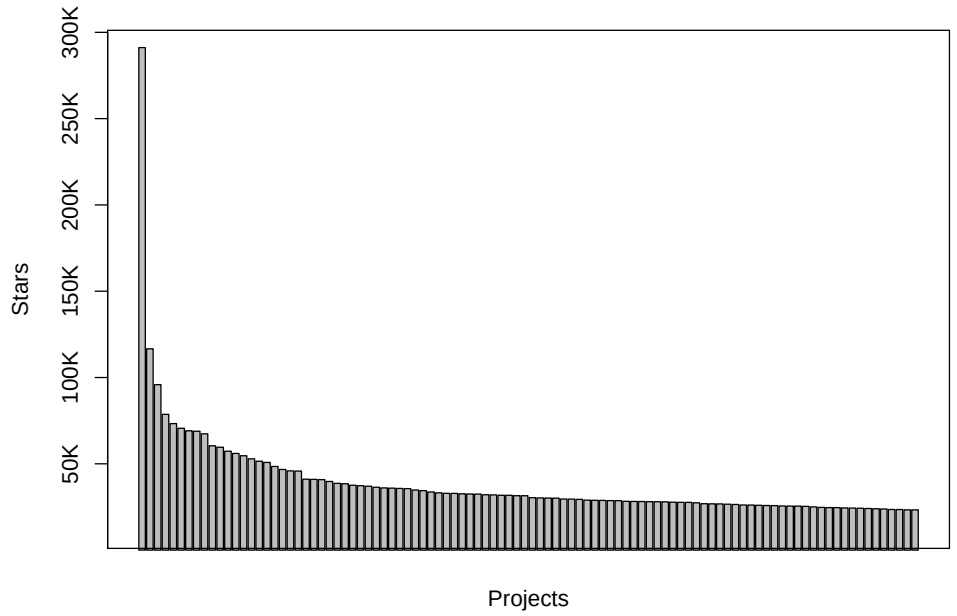


Figure 4.1: Number of GitHub stars of the analyzed projects

We navigated through the site pages, searching for more channels used to promote the projects.

After this manual inspection, the following promotion channels emerged:

- **Blogs**, which are used, for example, to publish announcements of new software versions, upcoming events, and improvements.
- **Events and Users Meetings:** Organizing events and supporting users meetings are other strategies commonly followed to promote projects. On events the initiative usually comes from the development team or from the organization that supports the project, whereas on user meeting the initiative comes from the users, usually from a specific region or country. We rely on Meetup[1] to discover users meetings.
- **Twitter, Facebook, and Google+**, which are also used to connect the projects to users. We considered only official accounts, which are explicitly advertised on the project documentation or are verified by the social network (e.g., `https://support.twitter.com/articles/20174631`).
- **Newsletter and RSS feeds**, which refer to e-mails with the most relevant news about the projects and RSS feeds.

[1] `https://meetup.com`

In addition, we found that developers use Q&A forums (e.g., StackOverflow), discussion groups (e.g., Google Groups), and messaging tools (e.g., IRC and Slack) to promote their projects. However, these channels are mostly used to discuss the projects and to provide answers to common questions raised by users. For example, from the 155 topics opened in 2017 in the ADOBE/BRACKETS discussion group at Google Groups, only eight (5.1%) are related to announcements of new versions, mostly pre-releases for community testing. Moreover, from almost 500 topics on FACEBOOK/REACT official forum, we could not identify any announcement related to the project development. Thus, in this study, we do not consider forums, discussion groups, and messaging tools as promotion channels.

4.3 Results

***RQ #1**: What are the most common promotion channels?*

Figure 4.2 presents the most common promotion channels used by the top-100 projects on GitHub. The most common channel is Twitter, which is used by 56 projects. The second one is Users Meetings (41 projects), followed by Blogs (38 projects), Events (33 projects), and RSS feeds (33 projects). The least common channels are Facebook and Google+, which are used by 18 and 7 projects, respectively.

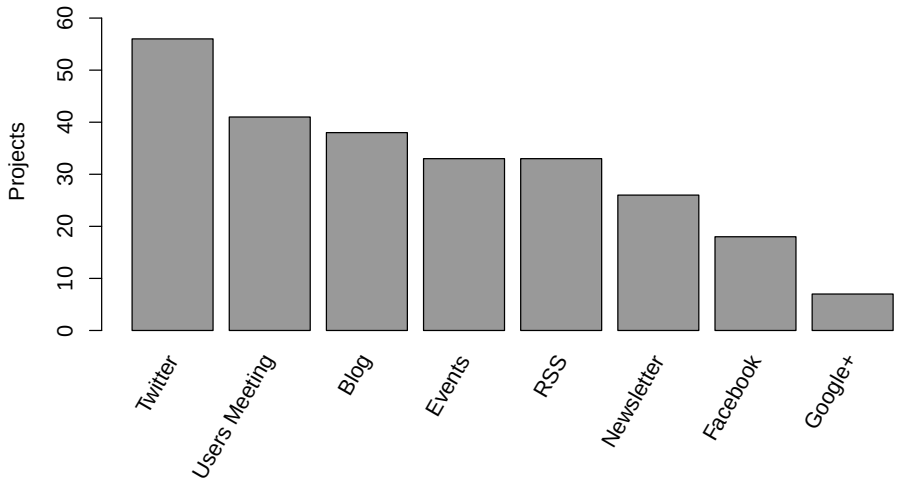


Figure 4.2: Most common promotion channels

Figure 4.3 shows the distribution of the number of promotion channels per project. Almost one third of the projects (32 projects) do not use any channel.

By contrast, more than half of the projects (55 projects) use at least two promotion channels. The highest number of promotion channels is seven, which is the case of facebook/react, facebook/react-native, meteor/meteor, golang/go, ionic-team/ionic, angular/angular, and adobe/brackets. We also found that Blog and Twitter is the most frequent combination of channels (35 projects). Other frequent combinations include, for example, Blog and RSS (31 projects), Events and Users Meetings (31 projects), and Twitter, Events and User Meetings (31 projects).

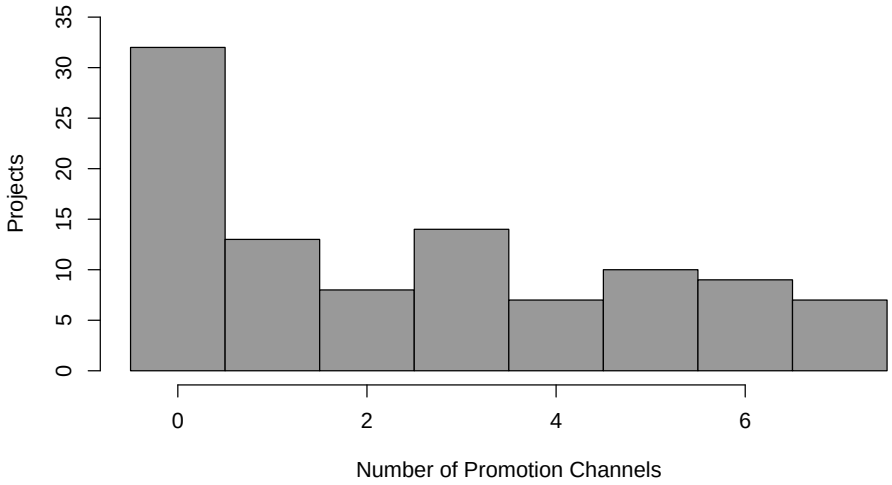


Figure 4.3: Number of promotion channels per project

Summary: Twitter is the most used promotion channel among the top-100 GitHub projects by stars. Moreover, most of the projects use at least two promotion channels. *Practical Implications:* Developers should prioritize Twitter as their main channel of communication with contributors and users.

***RQ** #2: How often do developers promote their projects?*

In this second question, we investigate how often developers promote their projects on blogs and social networks. For blogs, we calculate the promotion frequency as the number of posts on the last 12 months. For social networks, we could not retrieve all posts for all projects because their APIs restrict the search to a recent period (e.g., last seven days for Twitter and last 100 posts for Facebook). Thus, in this case, we only classified each social network account in two distinct groups: active and inactive. An *active* account has at least three posts on the last three months;

otherwise, it is considered an *inactive* account. This classification was performed by manually counting the number of posts on the social network pages.

Figure 4.4 presents the distribution of the number of blog posts on the last 12 months. The number ranges from 1 (NYLAS/NYLAS-MAIL) to 1,300 (FREECODECAMP/FREECODECAMP); the first, second, and third quartile values are 7, 19, and 54 posts, respectively.

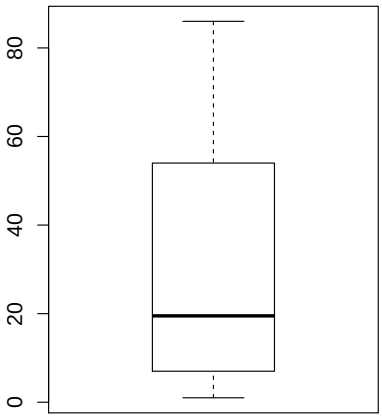


Figure 4.4: Distribution of the number of posts on the last 12 months (outliers are omitted)

Table 4.1 lists the activity status of the Twitter, Facebook, and Google+ accounts. We found that 83.9% of the projects that use Twitter have an active account; 55.6% of the projects have an active Facebook account and only 28.6% have an active Google+ account.

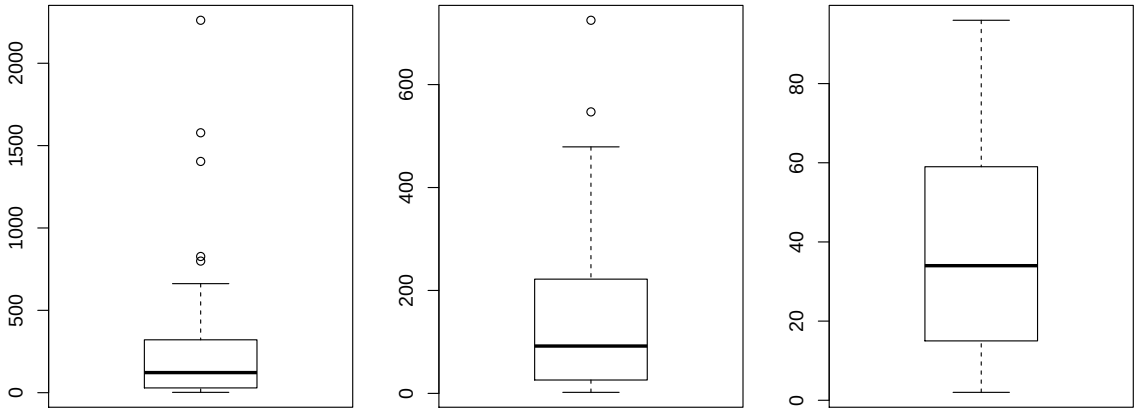
Table 4.1: Active Twitter, Facebook, and Google+ accounts

Channel	Active (%)	Inactive (%)
Twitter	47 (83.9%)	9 (16.1%)
Facebook	10 (55.6%)	8 (44.4%)
Google+	2 (28.6%)	5 (71.4%)

Finally, we investigate the characteristics of the user meeting groups promoted on Meetup (such meetings are the 3rd most common promotion channel studied in this article). A Meetup group is a local community of people that is responsible for organizing meeting events.[2] These groups are identified by topics to help members find them. Here, we rely on these topics to collect meetups about the studied open source projects, along with their locations (i.e., city and country). For example, the topic

[2]`https://www.meetup.com/help/article/902256`

for JQUERY/JQUERY is *jquery* and a summary of the meeting groups about this topic can be found at `https://www.meetup.com/topics/jquery/all`. Figure 4.5 presents the distribution of the number of groups, cities, and countries of the projects with meetings registered at Meetup. For groups, the values ranges from 2 to 2,261 groups; considering the cities, the values range from 2 to 725; finally, for countries, the values range from 2 to 96. The maximum values always refer to TORVALDS/LINUX. In other words, TORVALDS/LINUX has 2,261 meetup groups, which are spread over 725 cities from 96 countries.



(a) Groups (b) Cities (c) Countries

Figure 4.5: Number of groups, cities, and countries of the user meetings

Summary: Most projects post on their blogs with a high frequency (on median, 1.5 posts per month).

***RQ #3**: How popular and random projects differ on the usage of promotion channels?*

In the first research question, we investigated the most common promotion channels used by popular GitHub projects. In this section, we contrast the usage of promotion channels by these projects and by a random sample of GitHub projects. For this purpose, we randomly selected 100 projects from the top-5,000 repositories by number of stars and manually inspected their documentation using the same methodology reported in Section 4.2. The number of stars of this random sample ranges from 2,297 stars (UBER-ARCHIVE/IMAGE-DIFF) to 22,558 (VSOUZA/AWESOME-IOS).

Figure 4.6 compares the usage of promotion channels by the random projects and by the most popular ones. In the random sample, the number of projects using the

investigated promotion channels is significantly lower compared to the most popular ones. However, by applying the Spearman's rank correlation test, we found a strong correlation between the number of projects using the promotion channels on each group ($rho = 0.904$ and $p\text{-}value < 0.01$). For example, Twitter is also the most used promotion channel among the random projects (31 projects), followed by Blogs (17 projects) and RSS (13 projects). Compared to the most popular projects, Users meetings and Newsletter are less common (13 and 6 projects, respectively). Finally, Facebook and Google+ also have a very limited usage (7 and 4 projects, respectively).

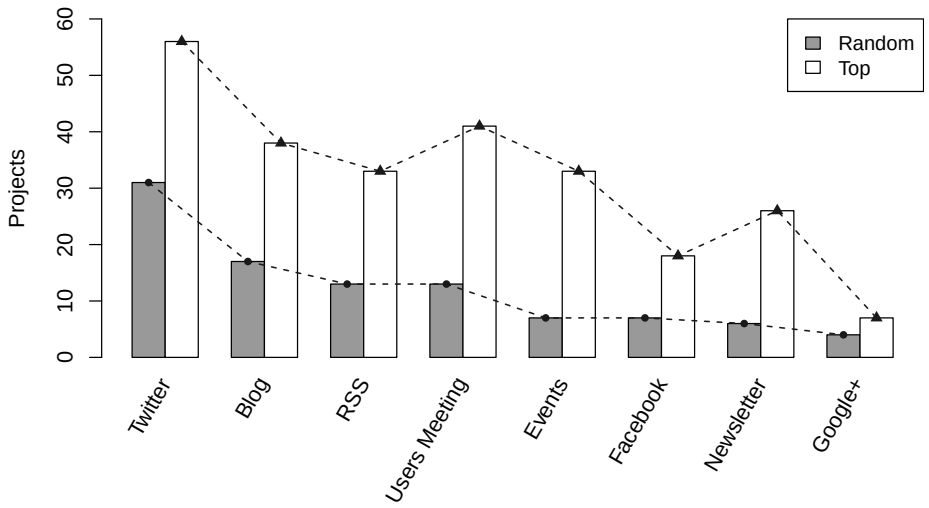


Figure 4.6: Most common promotion channels used by random projects

Summary: The number of projects using promotion channels is significantly lower among random projects.

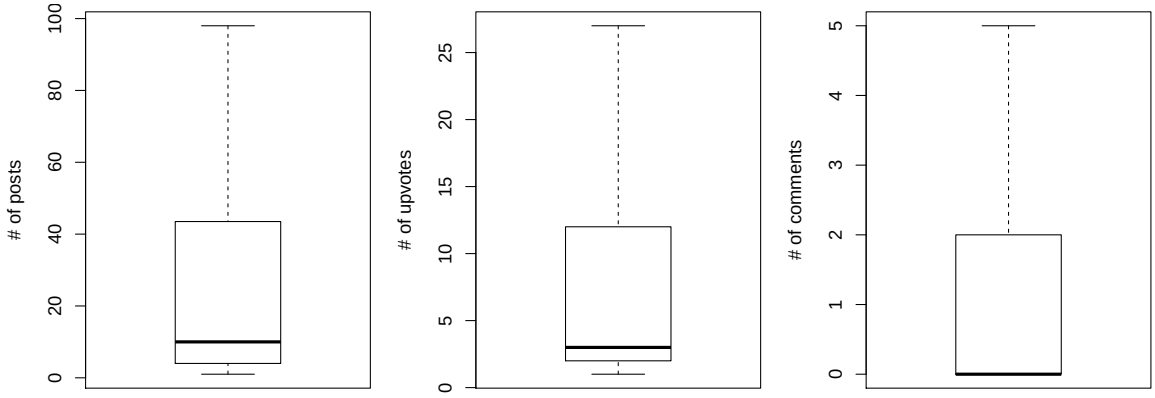
***RQ #4**: What is the impact of promotion on Hacker News?*

After publishing content on blogs, Twitter, etc., open source developers can also promote this content on social news aggregator sites. These sites aggregate contents from distinct sources for easing viewing by a large public. The most popular and important example is Hacker News,[3] which is dedicated to Computer Science and related technologies content. Hacker News posts just include a title and the URL of the promoted content (e.g., a blog post about a new version of an open source project). Any user registered in the site can post a link on Hacker News, i.e., not necessarily

[3] `https://news.ycombinator.com`

the links are posted by the contributors of an open source project, for example. Other Hacker News users can discuss the posts and upvote them. An upvote is similar to a *like* in social networks; posts are listed on Hacker News according to the number of upvotes. In this research question, we use Hacker News due to its popularity; posts that reach the front page of the site receive for example 10-100K page views, in one or two days.[4] Furthermore, Hacker News provides a public API, which allows search and metadata collection.

For each popular project considered in our study (100 projects), we searched for Hacker News posts with a URL referencing the project sites or pages, including GitHub pages (READMEs, issues, etc). As result, we found 3,019 posts on Hacker News referencing content from 96 studied projects (i.e., only four projects are never referenced on Hacker News). Figure 4.7 presents the distributions of the number of posts per project, upvotes, and comments. The number of posts ranges from 1 to 298 posts per project (RAILS/RAILS); the first, second, and third quartile values are 4, 10, and 43 posts, respectively. Regarding their upvotes, the most popular post is about APPPLE/SWIFT ("*Swift is Open Source*"), with 1,824 upvotes; the quartile values are 2, 3, and 12 upvotes, respectively. Finally, the highest number of comments is 760, about a GitHub issue opened for Microsoft Visual Studio ("*VS Code uses 13% CPU when idle due to blinking cursor rendering*"); the quartile values are 0, 0, and 2 comments, respectively. On the one hand, these results show that most Hacker News posts do not attract attention. By contrast, a small number of posts attract a lot of attention. For example, the top-10% posts have at least 132 upvotes. These posts are called *successful posts* in this investigation.



(a) Posts (b) Upvotes (c) Comments

Figure 4.7: Number of posts, upvotes, and comments (outliers are omitted)

Figure 4.8 shows boxplots with the number of GitHub stars gained by projects

[4] `https://goo.gl/evyP4w`

covered by successful posts, in the first three days before and after the publication date on Hacker News. The intention is to investigate the impact of a successful promotion on Hacker News, by comparing the number of stars gained before and after each successful post publication. On the median, the projects covered by successful posts gained 74 stars in the first three days before their appearance on Hacker News; in the first three days after the publication, the projects gained 138 stars. Therefore, Hacker News has a positive impact on the project's popularity, measured by GitHub stars. Indeed, the distributions are statistically different, according to the one-tailed variant of the Mann-Whitney U test (p-value ≤ 0.05). By computing Cliff's delta, we found a *medium* effect size ($d = -0.372$).

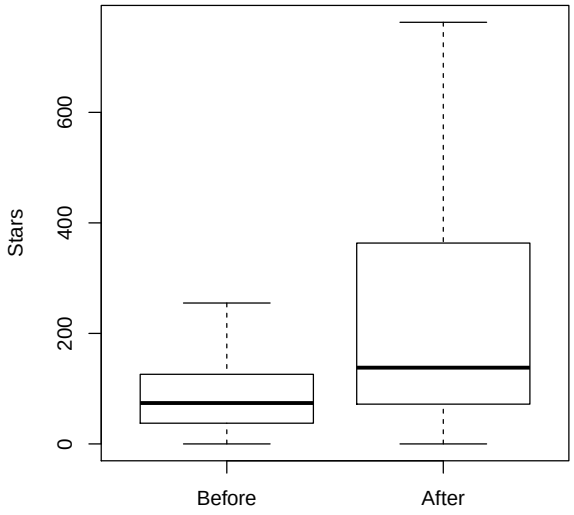


Figure 4.8: Number of GitHub stars received by projects covered by successful Hacker News posts in the first three days before and after the post publication

Finally, we inspected the titles of each successful post, aiming to categorize the post purpose. The most common category includes posts announcing new releases of open source projects (44.9%; e.g., "*Angular 2 Final Released*"). Other popular categories include posts promoting articles or reports about the projects (25.4%; e.g., "*Vue.js vs. React*"), announcing the first release of a project (16.5%; e.g., "*YouTube-dl: Open-source YouTube downloader*"), highlighting new project features (10.6%; e.g., "*Git and GitHub Integration Comes to Atom*") and open sourcing products (1.6%; e.g., "*Visual Studio Code is now open source*").

Summary: Most Hacker News posts do not attract attention, however the successful ones have a positive impact on the project's popularity. Moreover, most of the successful posts are related to new releases of open source projects. *Practical Implications:* GitHub developers should consider promoting new releases on news aggregator sites since it requires a low effort and can reach a large number of developers.

4.4 Threats to Validity

Dataset: In this study, we build an initial dataset with the top-100 repositories with most stars on GitHub. Thus, we can not generalize our results to all projects hosted on GitHub. To mitigate this point, we also analyzed a set of other 100 random repositories and the results were consistent to the top ones.

Documentation inspection: The analysis of the projects' documentations and websites were performed only by the author of this thesis. To mitigate this threat, the criteria to identify the channels were defined in agreement with a second researcher and the inspection was done with special attention.

Meetup topics: To analyze the characteristics of the user meeting, we relied on the topics provided by Meetup website. These topics are essentially useful since the website has more than 260.000 meeting groups spread in 178 countries. However, as they are added by the groups' organizers, there still the chance of some of them have been misconfigured or using non-representative topics.

4.5 Concluding Remarks

In this chapter, we investigated the most common promotion channels used by popular GitHub projects. First, we detailed the project selection and the inspection methodology used to collect the promotion channels used by the projects analyzed in this study. Next, we presented the most common promotion channels used by the top-100 projects on GitHub, by number of stars. We found that Twitter is the most common promotion channel, followed by Users Meetings, Blogs, Events, RSS Feeds, Newsletters, Facebook and Google+ (RQ #1). Then, we investigated how often developers

promote their projects on blogs and social networks. We showed that most projects posts on their Blogs with a high frequency (on median, 19 posts per year) and the large majority of the accounts on Twitter are active (i.e., has at least three posts on the last three months) (RQ #2). By contrasting the usage of promotion channels by popular projects and by random ones, we observed that the number of projects using the promotion channels is significantly lower among the random projects (RQ #3). Finally, we studied the impact of promoting projects' contents on Hacker News, which is nowadays the most popular news aggregator site. We showed that promotion has a relevant impact on the number of stars of the projects (RQ #4).

This study supports the following practical recommendations to open source project managers and leaders:

1. Promotion is an important aspect of open source project management, which should be emphasized by project leaders. For example, most popular GitHub projects (two thirds) use at least one promotion channel; half of the projects invest on two channels. By contrast, the use of promotion channels is less common among projects with lower popularity.

2. Open source project managers should consider the use of Twitter (47 projects among the top-100 most popular GitHub projects have active Twitter accounts), Users meetings (which are organized or supported by 41 projects), and blogs (which are used by 38 projects).

3. Open source project managers should also consider promotion on social news aggregator sites. Successful posts on Hacker News usually have a relevant impact on the popularity of GitHub projects. However, only 10% of the Hacker News posts about the studied projects have some success.

The data used in this study is publicly available at the Zenodo data repository: `https://doi.org/10.5281/zenodo.1226698`.

Chapter 5

Predicting the Popularity of GitHub Repositories

In this study, we extend our investigation presented in Chapter 3 by using of multiple linear regressions to *predict the popularity* of GitHub repositories. This chapter is organized as follows. First, we present the dataset used in the study (Section 5.3) and discuss the methodology followed in the study (Section 5.2). Section 5.4 presents our results, by exploring and discussing answers for the proposed research questions. Finally, Section 5.5 discusses threats to validity and Section 5.6 concludes the chapter.

5.1 Introduction

Prediction models have been successfully used to infer the popularity of content in other social networks, such as the number of views of YouTube videos [Roy et al., 2013; Pinto et al., 2013; Figueiredo, 2013] and the number of tweets associated to a given hashtag [Tsur and Rappoport, 2012; Ma et al., 2012, 2013]. However, to our knowledge, we are the first to attempt to predict the popularity—measured by the number of stars—of software projects hosted at GitHub. Specifically, we compute and investigate multiple linear regression models over two types of data: *generic* and *specific*. By generic, we refer to models produced from the complete dataset considered in this study, which includes historical data about the number of stars of 4,248 popular GitHub repositories. By specific, we refer to models produced from repositories that share similar growth trends. These trends are inferred using the KSC algorithm [Yang and Leskovec, 2011], which clusters time series with similar shapes. We address three major research questions in the study:

RQ #1: What is the accuracy of the generic prediction models? We report the Relative Squared Error (RSE) of the regression models computed using the time series of number of stars of all projects in our dataset.

RQ #2: What is the accuracy of the specific prediction models? First, using the KSC clustering algorithm, we identify four major growth trends among the systems in our dataset. Then, we evaluate the accuracy of the regression models computed over the time series of each cluster.

RQ #3: What is the accuracy of the repositories rank as predicted using the generic and specific models? In the previous RQs, our goal is to the predict the total number of stars, using generic and specific models. By contrast, in this final RQ, we evaluate the ability of both models to predict not the number of stars of a repository after a time, but its rank among the repositories in our dataset.

5.2 Study Design

In this section, we detail the techniques and models used to predict the number of stars of GitHub repositories. We also discuss how we evaluate the accuracy of these models.

Prediction Technique: We rely on multiple linear regression to predict the popularity of GitHub repositories. Multiple linear regression differs from simple regression by considering that all variables are not equally important [Freedman, 2009]. The general form of a multiple linear regression is as follows:

$$Y_t = b_0 + b_1 X_{t_1} + b_2 X_{t_2} + ... + b_r X_{t_r}$$

where Y_t is the dependent variable (number of stars at week t), X_{t_i} are the independent variables (stars in weeks i, $1 \leq i \leq r \leq t$), and b_j are the regression coefficients ($0 \leq j \leq r \leq t$).

Estimating the Errors: To evaluate the accuracy of the models, we use the Relative Squared Error (RSE). Assume that $N(r,t)$ is the real number of stars of a repository r in the week t. Moreover, assume that $\widehat{N}(r,t_r,t)$ is the number of stars *predicted* for r at the week t from the popularity data of the first t_r weeks. The RSE for this prediction

is given by [Pinto et al., 2013]:

$$RSE = \left(\frac{\widehat{N}(r, t_r, t)}{N(r, t)} - 1 \right)^2$$

For a collection $\mathcal{R}$ of repositories, the mean Relative Squared Error (mRSE) is defined as the arithmetic mean of the RSE values of all repositories in $\mathcal{R}$, as given by:

$$mRSE = \frac{1}{|\mathcal{R}|} * \sum_{r \in \mathcal{R}} \left(\frac{\widehat{N}(r, t_r, t)}{N(r, t)} - 1 \right)^2$$

Cross-Validation: As ilustrated in Figure 5.1, we perform cross-validation to assess the prediction models. We use 10 folds, i.e., the repositories are randomly partitioned in 10-folds and we use nine folds to build the prediction models (training set) and the remaining fold to evaluate their accuracy (validation set).

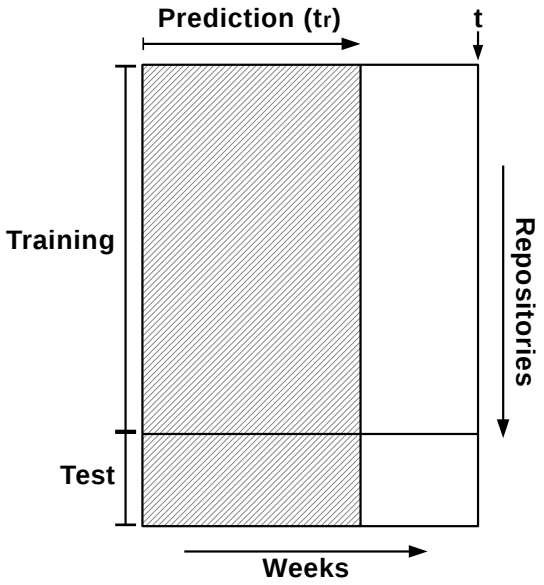


Figure 5.1: Cross Validation

Generic and Specific Models: We generate models for two datasets: *generic* and *specific*. By generic, we refer to models produced from the complete dataset, i.e., from the time series with the number of stars collected for 4,248 repositories. By specific, we refer to models produced from repositories that shared similar growth trends. As in our previous work [Borges et al., 2016], we rely on the KSC algorithm [Yang and Leskovec,

2011] to identify growth trends in our dataset. This algorithm clusters time series with similar shapes using a metric that is invariant to scaling and shifting. In other words, each cluster groups time series that share similar growth trends. Particularly, to answer RQ #2 we produce specific models considering only the time series in each cluster. We use the β_{CV} heuristic [Menasce and Almeida, 2001] to define the best number k of clusters. β_{CV} is defined as the ratio of the coefficient of variation of the intracluster distances and the coefficient of variation of the intercluster distances. The smallest value of k after which the β_{CV} ratio remains roughly stable should be selected. In our dataset, the values of β_{CV} stabilize for $k = 5$ (see Figure 5.2). Therefore, we configure KSC to produce five clusters.

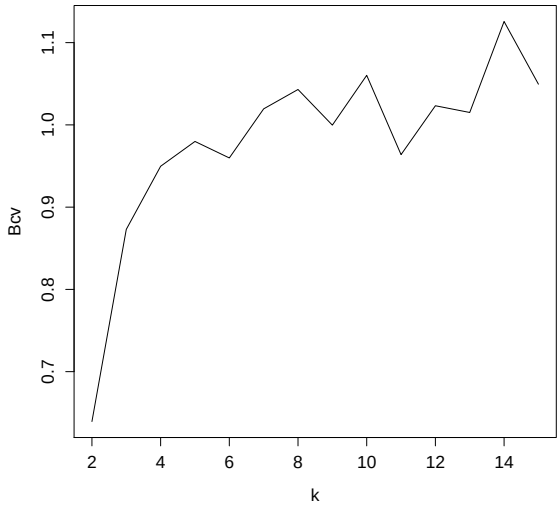


Figure 5.2: β_{CV} $(2 \leq k \leq 15)$

Figure 5.3 shows the time series representing the clusters' centroids of the five clusters. The trends presented by clusters C1, C2, and C3 suggest a linear growth in the number of stars. The trend presented by cluster C4 differs from the first three ones due variations in the number of stars over the time. Finally, cluster C5 suggests a viral growth.

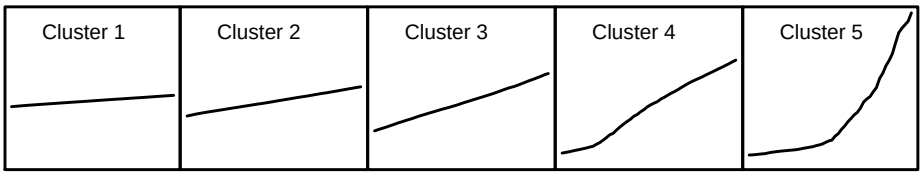


Figure 5.3: Five growth trends (clusters) identified for the repositories in our dataset

As presented in Table 5.1, cluster C1 concentrates almost half of the repositories in our dataset (49.1%) while cluster C5 has the lowest concentration (1.2%). Table 5.1 also presents the percentage of growth of each cluster, considering the centroids time series. This percentage ranges from 19.9% (cluster C1) to 1,659.1% (cluster C5).

Table 5.1: Popularity Trends Description

Cluster	# Repositories	% Growth
C1	2,087 (49.1%)	19.9
C2	1,456 (34.2%)	61.3
C3	521 (12.2%)	175.1
C4	131 (3.0%)	883.2
C5	53 (1.2%)	1,659.1

When answering RQ #2, we do not consider specific models for cluster C5 due to two main reasons: (a) it includes only 53 repositories (1.2%); (b) as presented in Figure 5.3, the time series in this cluster do not have a linear shape.

Repositories Ranking: To answer RQ #3, we compute three rankings: (i) repositories sorted according to the predicted number of stars using a generic prediction model (configured with $t_r = 26$ weeks and $t = 52$); (ii) repositories sorted according to the predicted number of stars using specific prediction models (configured with $t_r = 26$ weeks and $t = 52$); (iii) repositories sorted according to their real number of stars, as provided by GitHub API, on April 25, 2016, i.e., the last week we consider to build the time series of stars. In the first two rankings, the rank positions range from 1 to 4,248 (which is the dataset size). However, the third ranking includes all repositories in the previous rankings plus 468 repositories that entered the list of the most popular repositories in the year before April 25, 2016. As examples, we have apple/swift and Netflix/falcor. Netflix/falcor is not among the top-5,000 most popular repositories on April 25, 2015, when we select the repositories used in the study, but it gained popularity to the point of being the 481st most popular repository one year later. apple/swift was created on October 10, 2015; despite this it is the 23rd most popular repository on April 25, 2016, when we define the real ranking. In this way, the investigation conducted to answer RQ #3 includes the cases where a repository falls in the ranking not only due to a better performance of the repositories used to produce the prediction models, but also due to the performance of any other repository.

5.3 Dataset

The initial dataset used in this study includes historical data about the top-5,000 public repositories with more stars in GitHub. All data was obtained using the GitHub API, which provides services to search public repositories and to retrieve specific information about them (e.g., stars). First, we collect basic data about the repositories (i.e., owner, stars, creation date, programming language, etc.). Next, for each repository, we collect historical data about the number of stars. For this purpose, we used a service from the API that returns all star events of a given repository. For each star, these events store the date and the user responsible to starring the repository. However, GitHub API returns at most 100 events by request (i.e., a page) and at most 400 pages. For this reason, it is not currently possible to retrieve all star events of systems with more than 40K stars, which is the case of seven repositories: FREECODECAMP (112,397 stars), TWBS/BOOTSTRAP (95,293 stars), VHF/FREE-PROGRAMMING-BOOKS (54,208 stars), MBOSTOCK/D3 (49,173 stars), ANGULAR/ANGULAR.JS (48,787 stars), FORTAWESOME/FONT-AWESOME (41,621 stars), FACEBOOK/REACT (41,037 stars). Moreover, 278 repositories have no main programming language identified. These repositories do not store source code, e.g., JLEVY/THE-ART-OF-COMMAND-LINE (26,298 stars) or are moved/removed repositories, e.g., NODEJS/NODE-V0.X-ARCHIVE (37,354 stars). Therefore, we also remove these repositories from the dataset. Additionally, we only consider the stars gained in the last 52 weeks of each repository. Thus, repositories with less than 52 weeks are also removed from the dataset (468 repositories).

Figure 5.4 shows the number of stars of the 4,248 repositories in our dataset. This number ranges from 39,149 stars (JQUERY/ JQUERY) to 1,248 stars (MIKEFLYNN/EGG.JS). As presented, the distribution is right skewed (quantiles 5% = 1,307 stars and 95% = 9,360 stars). The mean and median number of stars are 3,393 and 2,240, respectively. Table 5.2 lists the top-10 repositories with more stars. These repositories have at least 30K stars and belong to four different domains (Web Frameworks and Libraries, Software Tools, Documentation, and System Software).

Next, we built the stars time series of each repository from the stars events. These time series consist of the number of stars gained by week since the repository creation date up to April 25, 2016, when we collected our data. As an example, Figure 5.5 shows the time series retrieved for JQUERY/JQUERY, the most starred repository in our dataset. This repository has 369 weeks (x-axis) and the number of stars increased from 1,692 stars to 39,149 stars (y-axis).

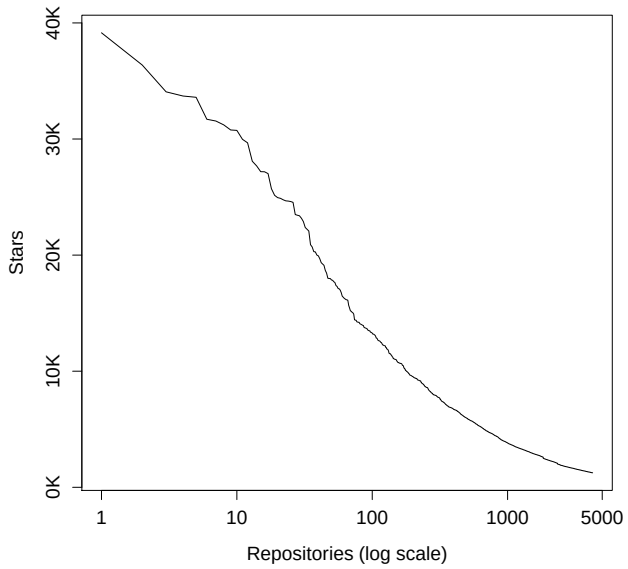


Figure 5.4: Repositories popularity

Table 5.2: Top-10 repositories with more stars

Repository	Domain	# Stars
jquery/jquery	Web	39,149
robbyrussell/oh-my-zsh	Tools	36,373
airbnb/javascript	Doc	34,064
h5bp/html5-boilerplate	Web	33,704
meteor/meteor	Web	33,594
torvalds/linux	System	31,702
daneden/animate.css	Web	31,549
facebook/react-native	Web	31,217
rails/rails	Web	30,779
docker/docker	System	30,742

5.4 Results

***RQ #1**: What is the accuracy of the generic prediction models?*

In order to start answering this question, we produce *generic* prediction models and assess their accuracy using 10-fold cross validation for different values of t_r (prediction data, see Figure 5.1). In all cases, we use the models to predict the number of stars at week 52 ($t = 52$, in Figure 5.1). In other words, we use the number of stars in the first t_r weeks to predict the number of stars in the 52nd week (last week we

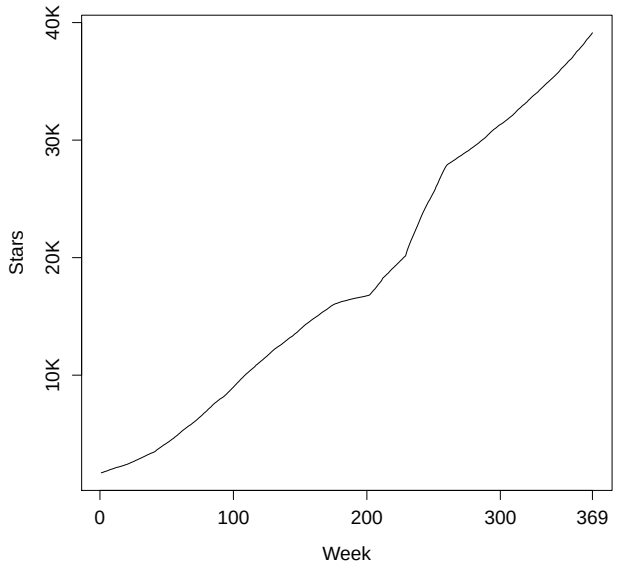


Figure 5.5: JQUERY/JQUERY time series (369 weeks)

considered when collecting the number of stars). Figure 5.6 reports the average error (mRSE) across all models.

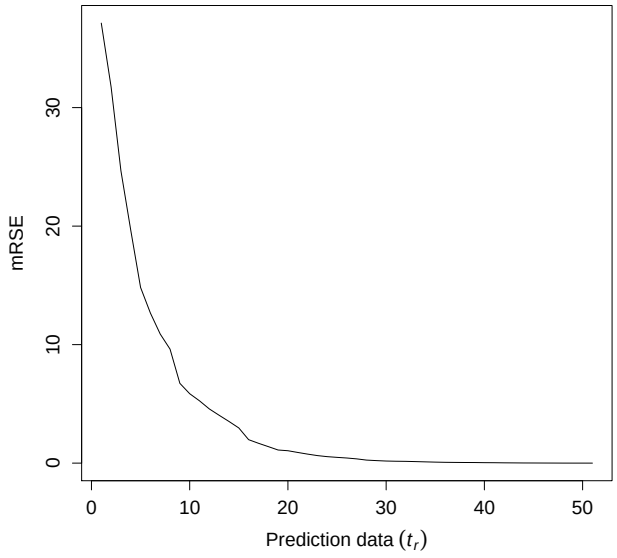


Figure 5.6: Generic model error

For small values of t_r the models do not perform well, e.g., for $t_r = 10$ weeks mRSE $= 5.858 \pm 4.372$ (mean $\pm$ 95% confidence interval). However, as we increase the values of t_r, the results are more accurate. For example, mRSE $= 0.432 \pm 0.257$ for

$t_r = 26$ weeks. This means that we can predict with a low error the number of stars six months ahead, using as training data the past six months of stars.

Figure 5.7 shows a scatter plot that correlates the number of stars gained and the RSE for the generic models produced using $t_r = 26$ weeks. Each point in this figure represents a repository. We ran Spearman's rank correlation test and the resulting correlation coefficient *rho* is -0.50, with *p-value* < 0.001. Therefore, the generic models are more accurate for the repositories that gained many stars in the period.

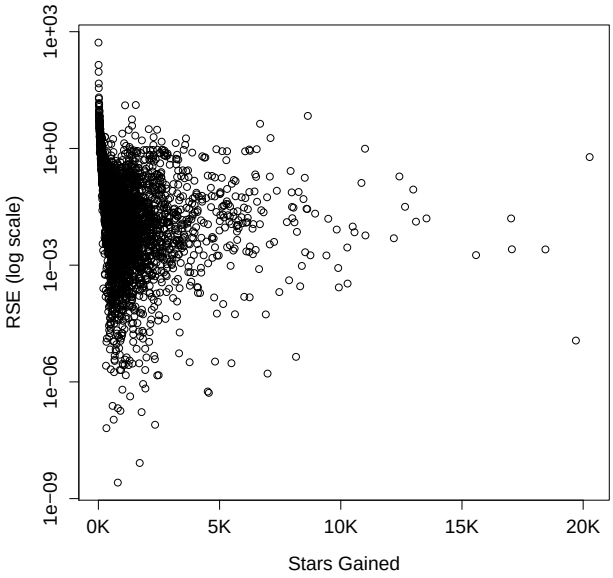


Figure 5.7: Stars vs RSE (generic model, $t_r = 26$ weeks)

Table 5.3 lists the prediction results for the top-10 and bottom-10 repositories with more stars in our dataset. The column "Stars" shows the real number of stars gained in 52 weeks and the column "Predicted" presents the number of stars predicted for the same period using a generic model ($t_r = 26$ weeks and $t = 52$). The difference between the real and the predicted values ranges from 1.64% to 12.84%, in absolute values, for the top-10 repositories and from 2.83% to 100% for the bottom-10 ones.

Finally, we evaluate the accuracy of the generic prediction models for different values of the target week t. Figure 5.8 shows the average error for $t = 26$ weeks (half year) and $t = 104$ weeks (two years). The figure also includes the average error for $t = 52$ weeks (already presented in Figure 5.6). In all cases, we see a decreasing trend of the average error measure. However, for higher values of t, we need less prediction data to achieve similar average errors. For example, using $t = 52$ weeks and a fraction of time equals to 0.5 (i.e, 26 weeks) the average error (mRSE) is 0.432 ± 0.257. For $t = 104$ weeks, a similar average error (mRSE $= 0.460 \pm 0.182$) happens for a fraction of time of 0.36 (i.e., 38 weeks).

Table 5.3: Number of stars gained (real and predicted measures) for the top-10 (first table half) and bottom-10 repositories (second table half). Predictions are produced using a generic model ($t_r = 26$ weeks and $t = 52$). We can see that the error (column "% Diff") is lower for the top-repositories.

Repository	Stars	Predicted	% Diff
jquery/jquery	6,160	5,369	-12.84
robbyrussell/oh-my-zsh	13,536	11,829	-12.61
airbnb/javascript	17,026	14,882	-12.59
h5bp/html5-boilerplate	4,896	4,691	-4.19
meteor/meteor	9,919	10,082	+1.64
torvalds/linux	10,566	9,682	-8.37
daneden/animate.css	10,492	9,452	-9.91
facebook/react-native	18,443	19,373	+5.04
rails/rails	5,701	5,128	-10.05
docker/docker	10,268	9,721	-5.33
ReactiveRaven/jqBootstrapValidation	213	298	+39.91
infinitered/ProMotion	119	238	+100.00
nslocum/design-patterns-in-ruby	640	731	+14.22
jbt/markdown-editor	621	744	+19.81
mumble-voip/mumble	565	667	+18.05
Manabu-GT/ExpandableTextView	676	623	-7.84
apache/flink	890	712	-20.00
mafintosh/mongojs	322	381	+18.32
rofl0r/proxychains-ng	813	790	-2.83
mikeflynn/egg.js	584	793	+35.79

Summary: The generic models start to provide accurate predictions when they are trained with data from six months and used to predict the number of stars six months ahead. Furthermore, generic models for highly popular repositories are more accurate than the ones generated for repositories with few stars.

RQ #2: *What is the accuracy of the specific prediction models?*

In this second research question, we generate *specific* prediction models for the repositories in each cluster (presented in Section 5.2) and assess their accuracy using 10-fold cross validation for different values of t_r. As in the first question, we predict the number of stars at week 52.

Figure 5.9 reports the average error across all specific models. Cluster C1, which concentrates almost half of the repositories, presents a fast decreasing in the average error, e.g., mRSE = 18.500 ± 14.501 for $t_r = 1$ week and 0.127 ± 0.020 for $t_r = 10$ weeks.

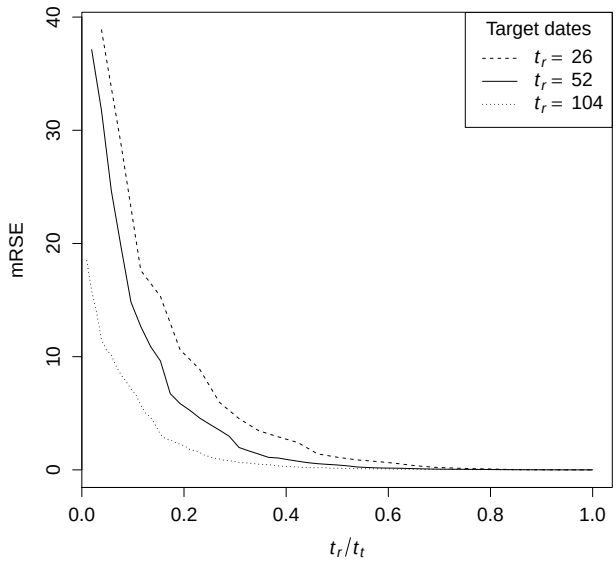


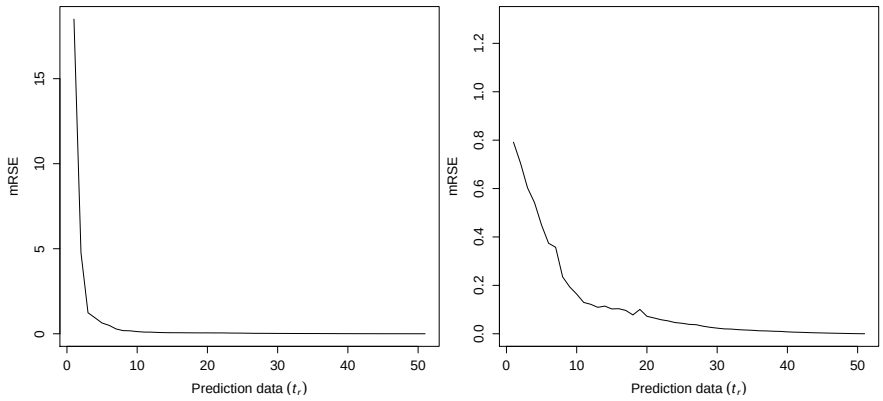
Figure 5.8: Generic model error (y-axis). Predictions for 26, 52, and 104 weeks, using different fractions of data (x-axis)

This suggests that specific models for this cluster require very few data to provide accurate predictions. Cluster C2 also presents accurate results for any value of t_r. As we can observe, the accuracy of the models for cluster C2 is higher than the accuracy for C1 when considering $t_r \leq 6$ weeks. However, for $t_r > 6$ weeks, the accuracy of C2 is slightly lower. For example, mRSE $= 0.030 \pm 0.009$ ($t_r = 26$ weeks) and mRSE $= 0.038 \pm 0.009$ ($t_r = 26$ weeks) for clusters C1 and C2, respectively. Cluster C3, which presents the fastest linear trend, shows an initial increasing in the average error, followed by a drastic reduction. This happens due to inaccurate results of two repositories: TESSALT/ECHO-CHAMBER-JS (RSE = 284.29) and GILESBOWKETT/REWIND (RSE = 224.48), which gained a high number of stars at weeks 8 and 13, respectively.[1]

Figure 5.10 shows boxplots with the improvements per cluster. The improvements are calculated from the gains achieved by specific models ($t_r = 26$ weeks). As we can observe, specific models improve the predictions in all clusters, considering the median values. The median improvements for each cluster are 15.72%, 1.08%, 2.00%, and 6.66%, respectively. The repositories in cluster C1 take more advantage of specific models (1st quartile = 2.43%). By contrast, clusters C3 and C4 have the highest percentage of repositories with a worst performance (1st quartile equal to -9.46% and -11.79%, respectively).

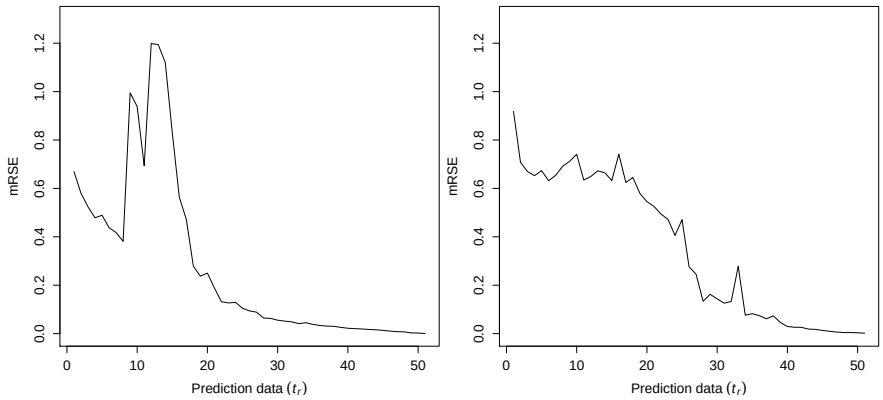
Table 5.4 lists the specific prediction results for the top-10 and bottom-10 repositories with more stars in our dataset. The column "Stars" shows the real number

[1]Because cluster C5 does not follow a linear trend it is not included in our analysis.



(a) Cluster C1

(b) Cluster C2



(c) Cluster C3

(d) Cluster C4

Figure 5.9: Model prediction error for different growth trends (i.e., clusters extracted using the KSC algorithm)

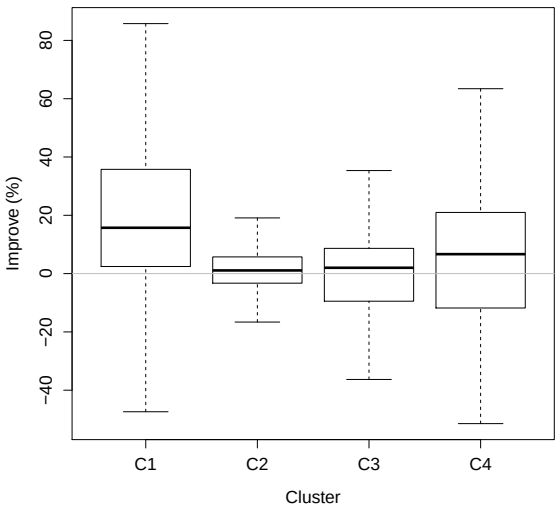


Figure 5.10: Improvement of specific models per cluster (outliers are omitted).

of stars gained in 52 weeks and the column "Predicted" presents the number of stars predicted for the same period using specific models ($t_r = 26$ weeks). The difference between the real and predicted number of stars ranges from 0.06% to 18.29%, in absolute values, for the top-10 repositories. Column "% Improve" shows the gains achieved by specific models, when compared with the predictions provided by the generic models. As we can see, the specific models increase the accuracy of the predictions for six out of ten repositories, from 3.39% to 7.72%. For the bottom-10 repositories, the difference between the real and predicted values ranges from 1.88% to 44.06% (column "% Diff"). In this case, the specific models produced more accurate results for seven out of ten repositories (column "% Improve").

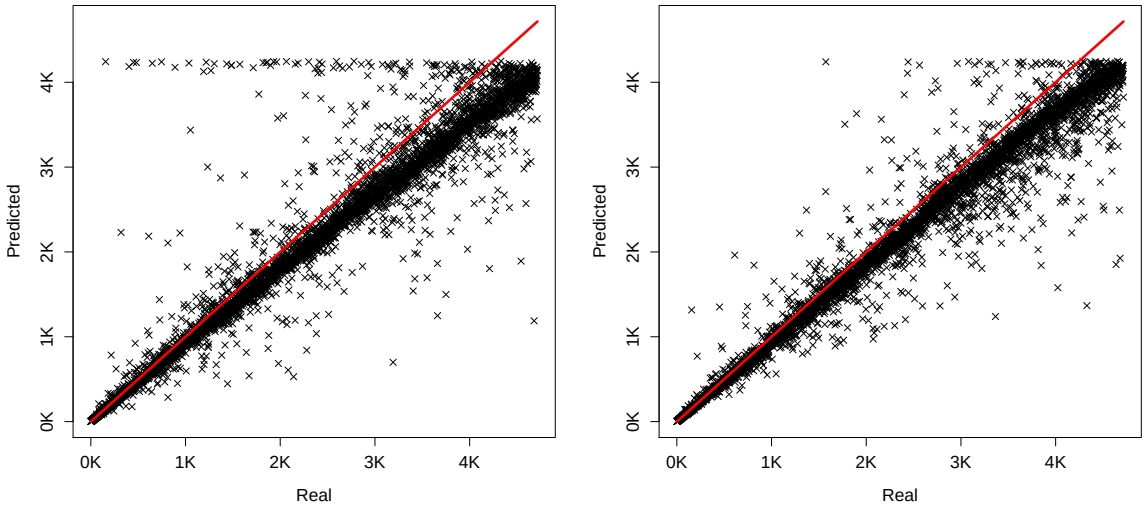
Table 5.4: Number of stars gained (real and predicted measures) for the top-10 (first table half) and bottom-10 repositories (second table half). Predictions are produced using specific models ($t_r = 26$ weeks and $t =$ week 52). Column "% Improve" shows the gains achieved by specific models, when compared with the predictions provided by generic models. Black bars represent positive gains and gray bars denote negative gains.

Repository	Cluster	Stars	Predicted	% Diff	% Improve
jquery/jquery	C1	6,160	5,578	-9.45	+3.39
robbyrussell/oh-my-zsh	C2	13,536	12,826	-5.25	+7.37
airbnb/javascript	C2	17,026	20,140	+18.29	-5.70
h5bp/html5-boilerplate	C1	4,896	4,690	-4.21	-0.02
meteor/meteor	C2	9,919	10,571	+6.57	-4.93
torvalds/linux	C2	10,566	10,498	-0.64	+7.72
daneden/animate.css	C2	10,492	10,045	-4.26	+5.65
facebook/react-native	C3	18,443	18,432	-0.06	+4.98
rails/rails	C1	5,701	5,386	-5.53	+4.53
docker/docker	C2	10,268	9,468	-7.79	-2.46
ReactiveRaven/jqBootstrapValidation	C1	213	209	-1.88	+38.03
infinitered/ProMotion	C1	119	155	+30.25	+69.75
nslocum/design-patterns-in-ruby	C3	640	922	+44.06	-29.84
jbt/markdown-editor	C2	621	667	+7.41	+12.40
mumble-voip/mumble	C2	565	583	+3.19	+82.35
Manabu-GT/ExpandableTextView	C3	676	729	+7.84	0
apache/flink	C3	890	853	-4.04	+14.29
mafintosh/mongojs	C1	322	309	-4.04	+14.29
rofl0r/proxychains-ng	C3	813	886	+8.98	-6.15
mikeflynn/egg.js	C1	584	523	-10.45	+25.34

Summary: Repositories in cluster C1 (slow growth, 49.1% of the repositories) demand less data to produce reliable predictions. Cluster C1 has also the highest percentage of repositories taking advantage of specific models. Furthermore, specific models improved the predictions of six (out of ten) top systems, from 3.39% to 7.72%. They also improved the predictions of seven (out of ten) bottom systems, from 1.88% to 44.06%. Therefore, specific models are recommended for repositories with slow growth (cluster C1) and/or among the ones with less stars.

***RQ #3**: What is the accuracy of the repositories rank as predicted using the generic and specific models?*

Figure 5.11 shows scatter plots correlating the real rank and predicted rankings using generic and specific models. The red line represents the identity function, i.e., a perfect match between the real and predicted ranks. Points above this line are repositories where the predicted rank is higher than the real one (we refer to this kind of error as an underestimation; e.g., a repository is predicted at the 10th position, but in fact it is in position 5th). By contrast, points below the identity line have a predicted rank lower than the real one (we refer to this error as an overestimation; e.g., a repository is predicted at the 5th position, but in fact it in in position 10th). Initially, we can observe that both models tend to overestimate many predictions, i.e., we usually have more points below the identity line. This happened because 468 repositories were created and/or quickly became more popular than the ones in our dataset. These repositories are called newcomers, in the context of this research question. Suppose for example that a newcomer appears at rank i; in this case it increases the rankings of all repositories with a predicted rank greater than i. This shift in the rankings is not detected by the prediction models we investigate, since they do not have information about new systems appearing in the rankings. However, we decide to consider newcomers in this first part of RQ #3 to simulate a situation that will appear in the practice.



(a) Using generic models (b) Using specific models

Figure 5.11: Real vs predicted rankings. The red line is the identity function.

To compare the real and the predicted rankings, we use the Spearman's correlation test. Since the test requires as input two vectors with the same size, we removed the newcomers from the real rankings. For the generic model, we found a strong correlation between the ranks ($rho = 0.9534$ and $p\text{-}value < 0.001$). For the specific models, the correlation is slightly higher ($rho = 0.9777$ and $p\text{-}value < 0.001$). Figure 5.12 shows the Spearman's coefficient for different groups of top-repositories. For the top-16 repositories, the correlation using the generic model is lower than the one using the specific models ($rho = 0.9321$ and $rho = 0.9821$, respectively). For the other top-values values, this difference decreases. However, the rankings as predicted by the specific models present slightly higher results in all cases.

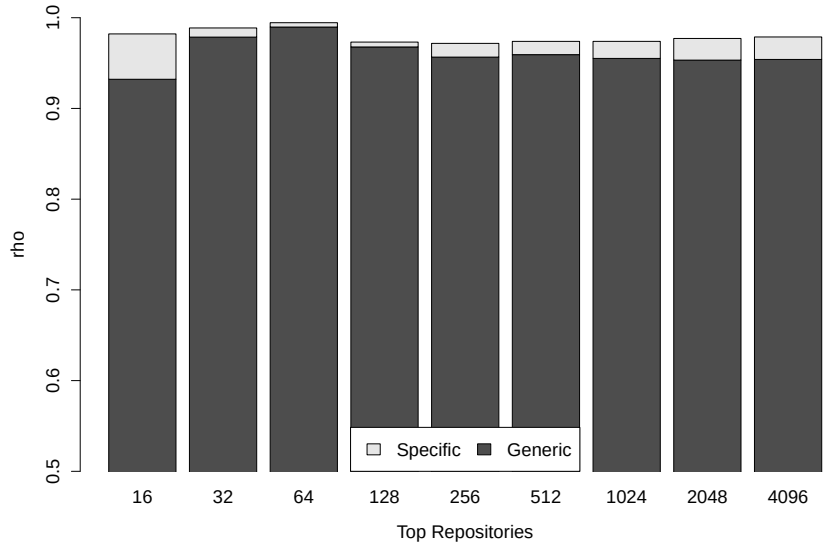


Figure 5.12: Spearman's rank correlation *rho* between predicted and real rankings per group of top-repositories ($p\text{-}value < 0.001$).

Table 5.5 shows the predicted rank for the top-10 and bottom-10 repositories in our dataset. The column "Real" represents the real rank in GitHub when our dataset was collected. The column "Predicted" shows the predicted rank using the generic (column "Generic") and the specific models (column "Specific"). Repositories that were created or became popular after the date we start collecting the time series are marked with "—". Finally, the column "Diff" shows the difference between the predicted and the real rankings. As mentioned, both predictions are more accurate for the top repositories and tend to overrate the rank of the bottom repositories. For the top-10 repositories, the difference in absolute values between the ranks ranges from 0 to 2 (generic models) and from 0 to 1 (specific models). For the bottom-10 repositories, the

difference between the ranks ranges from 528 to 1,145 (generic models) and from 520 to 892 (specific models). However, it is important to notice that the distribution of the number of stars per repository is heavy-tailed Borges et al. [2015]. Therefore, minor differences in the predicted number of stars can represent a movement of hundreds of positions in the relative order of a repository. For example, an error of 175 stars in the number of stars predicted to APACHE/FLINK is responsible to generate a error of 528 positions in its ranking (from position 4,708 in the real ranking; to position 4,180 in the ranking predicted using a generic model).

Table 5.5: Real and predicted rankings for the top-10 (first table half) and bottom-10 repositories (second table half), using the generic and the specific models. Marks "—" indicate repositories that were created and/or became popular after the date we set to select the repositories considered in this study.

Repository	Real	Predicted		Diff	
		Generic	Specific	Generic	Specific
jquery/jquery	1	1	1	0	0
robbyrussell/oh-my-zsh	2	2	2	0	0
airbnb/javascript	3	5	3	2	0
h5bp/html5-boilerplate	4	4	5	0	1
meteor/meteor	5	3	4	-2	-1
torvalds/linux	6	8	6	2	0
daneden/animate.css	7	9	8	2	1
facebook/react-native	8	6	7	-2	-1
rails/rails	9	10	10	1	1
docker/docker	10	11	11	1	1
jbt/markdown-editor	4,707	3,908	4,001	-799	-706
apache/flink	4,708	4,180	4,167	-528	-541
google/ion	4,709	—	—	—	—
Manabu-GT/ExpandableTextView	4,710	4,149	4,016	-561	-694
iPaulPro/Android-ItemTouchHelper-Demo	4,711	—	—	—	—
mumble-voip/mumble	4,712	4,011	4,092	-701	-620
mafintosh/mongojs	4,713	4,080	4,164	-633	-549
mikeflynn/egg.js	4,714	3,569	4,194	-1,145	-520
wequick/Small	4,715	—	—	—	—
rofl0r/proxychains-ng	4,716	4,136	3,824	-580	-892

Summary: Prediction models tend to overestimate repositories ranks, specifically due to the entry of newcomers in the list of popular repositories. However, when newcomers are not considered, there is a strong correlation between predicted and real rankings (*rho* = 0.9534 and 0.9777, for generic and specific models, respectively.)

5.5 Threats to Validity

Measuring popularity using the number of stars. In the investigation reported in this study, we measure popularity using the number of stars of the GitHub repositories, as in other studies [Weber and Luo, 2014; Aggarwal et al., 2014]. However, we highlight that developers can star a repository for other reasons, for example, to create bookmarks.

Unstar. In GitHub, users can unstar a repository that they starred before. As GitHub does not provide data about these events, we do not take into account these events in our study.

Repositories selection. GitHub has 17,136,765 public repositories, including forks (in June 15, 2016). For this study, we started with the top-5,000 repositories with more stars and after a cleaning step, we analyze 4,248 repositories. However, we stress that our goal is to predict popularity of most starred repositories. For example, 12,462,551 repositories (73%) have no stars and probably will never receive one in their lifetime. In other words, it is probably easier (and less useful) to make predictions for a dataset with all public repositories in GitHub.

Growth trends. The selection of the number of clusters is a key parameter in clustering algorithms like KSC. To mitigate this threat, we use the β_{CV} heuristic [Menasce and Almeida, 2001] to define the best number k of clusters. We also discard cluster C5 from our evaluation of specific models (RQ #2), since the repositories in this cluster do not follow a linear growth. Notice, however, that cluster C5 includes only 53 repositories (1.2%).

5.6 Concluding Remarks

In this study, we use multiple linear regressions to predict the popularity of GitHub repositories. We found that general models, i.e., models produced using the top GitHub repositories, start to provide accurate predictions when they are trained with data from six months and used to predict the number of stars six months ahead (RQ #1). We also found that specific models, i.e., models produced using repositories that share the same growth trend, can reduce the average prediction error and produce reliable predictions using less data. For the most common growth trend in our dataset, which includes almost half of the repositories, specific models improved significantly the accuracy of the predictions (RQ #2). Finally, we report that prediction models tend to overestimate

the repositories ranks. However, when newcomers are not considered, there is a very strong correlation between predicted and real rankings (RQ #3).

Chapter 6

Conclusion

In this chapter, we present our closing points and arguments. We summarize the key findings of this thesis (Section 6.1) and review our main contributions (Section 6.2). Next, in Section 6.3, we discuss other implications of our findings. Finally, we outline possible ideas for future work (Section 6.4).

6.1 Summary

Social coding platforms are disrupting the way developers collaborate on open source software development. In addition to version control systems, modern social coding platforms usually integrate features typical of social networks. For example, inspired by the *like* button of such networks, GitHub users can *star* a repository. However, the real and practical meaning of "starring a project" was never the subject of an in-depth and well-founded empirical investigation. For example, in a preliminary study with 400 StackOverflow users, we found that stars are viewed by practitioners as the most useful measure of popularity on GitHub. Considering this context, in this thesis, we propose and evaluate—through a set of quantitative and qualitative studies—practical guidelines and insights to help project managers understand and improve the popularity of their projects.

First, as result of the study reported in Chapter 3, we confirmed that stars are a reliable popularity measure; therefore, project managers should track and compare the number of stars of their projects with competitor ones. We collected historical data about the number of stars of 5,000 popular GitHub repositories. We used this dataset to characterize the main factors that impact the number of stars of these projects. We also proposed, characterized, and validated four common popularity growth patterns,

which were derived after clustering the time series that describe the number of stars of the projects. We summarize our findings in this chapter as follows.

- 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%). Particularly, three out of four developers consider the number of stars before using or contributing to GitHub projects.

- JavaScript is the language with the highest number of popular repositories (median of 3,163 stars). The top-3 most popular application domains are (1) systems software, (2) applications, and (3) web libraries and frameworks. Repositories owned by organizations tend to be more popular than the ones owned by individuals. We did not find a correlation between numbers of stars and the repository's age; however, we found a low correlation with commits, and a moderate correlation with contributors and forks.

- Unstar events (i.e., when users remove their stars) do not have a major influence on the popularity of GitHub repositories. Furthermore, projects containing documentation are those who developers lose their interest faster. Finally, we found that repositories have a tendency to receive more stars right after their first public release. After this period, the growth rate tends to stabilize.

- Slow growth is the dominant growth pattern, including 58.2% of the repositories in our dataset. Then, moderate growth is the second pattern with more repositories (30.0%), followed by fast growth (9.3%). Finally, we show that viral growth includes repositories with a massive growth in their number of stars. However, it is a less common pattern, including 2.3% of the repositories.

- *Age* is the most influential feature to discriminate repositories in the proposed growth patterns. The second and third most influential features are number of *Issues* and *Last Push*, respectively. 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.

- According to developers of the studied projects, 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 technologies*, and *innovative project*. Finally, the major reason for viral growth is also *promotion on social media sites* (72.7%).

In order to increase the chances of long-term success, it is very important to continually attract more participants and contributors to open source projects. In Chapter 4, we investigated the most common promotion channels used by the top-100 projects on GitHub, by number of stars. We concluded that promotion is an important aspect of open source project management, which should be emphasized by project leaders. For example, most popular GitHub projects (two thirds) use at least one promotion channel; half of the projects invest on two channels. We also contrasted the usage of promotion channels by popular projects and by random ones. We observed that the number of projects using the promotion channels is significantly lower among the random projects. Finally, we studied the impact of promoting projects on popular news aggregator site, particularly on Hacker News. We showed that successful posts usually have a relevant impact on the popularity of GitHub projects. However, only 10% of the Hacker News posts have some success.

Finally, in Chapter 5, we studied the use multiple linear regressions to predict the popularity of GitHub repositories. Specifically, we computed and investigated the use of generic prediction models, which are the ones produced using all projects in the dataset, and the use of specific prediction models, which are the ones produced from repositories sharing the same growth pattern. We found that general models start to provide accurate predictions when they are trained with data from six months and used to predict the number of stars six months ahead. Moreover, specific models improve significantly the accuracy of the predictions and reduce the amount of data needed to build the prediction models. Finally, we evaluated the ability of the prediction models to predict the rank of a repository after a time. We reported that the proposed prediction models tend to overestimate the repositories ranks; however, when newcomers are not considered, there is a strong correlation between predicted and real rankings.

6.2 Contributions

We summarize our contributions as follows:

- We reveal the developers **motivations for starring GitHub projects** through a large-scale investigation with 791 participants. We also reveal that developers star repositories mainly to show appreciation to projects. We also reveal that

three-quarters of the participants consider the number of stars before using or contributing to a project.

- We quantitatively **characterize the popularity of a large set of 5,000 GitHub repositories**. We show how popularity varies per programming language, application domain, and repository owner. We correlate popularity with other characteristics of a repository, like age, number of commits, number of contributors, and number of forks. Finally, we reveal that new features have a major impact on popularity.

- We propose **four patterns of popularity growth**: *Slow*, *Moderate*, *Fast*, and *Viral*. We also identify endogenous factors that distinguish the repositories in each growth pattern (e.g., *frequent updates*, *development history*, and *frequent releases*). Moreover, we reveal the perceptions of repositories' owners on these patterns.

- We address the **most common channels used by developers to promote open source projects**. We list the most common promotion channels and show how often developers promote their projects on those channels. We differ the usage of promotion channels among popular and random projects and show the impact of promotion on a modern social news aggregator site.

- We propose and evaluate the use **multiple linear regression models to predict the popularity** of GitHub repositories. We show that generic prediction models start to provide accurate predictions when they are trained with data from six months and used to predict the number of stars six months ahead. We also show that specific prediction models can reduce the average prediction error and reduce the amount of data required to provide predictions.

- We developed **an open-source supporting website**, called GitTrends.io (`http://gittrends.io`), that developers can use to visualize the popularity history of GitHub projects, compare the repositories growth with other ones, in order to support their decisions on use open source projects.

- 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.

6.3 Key Findings and Discussion

In Chapter 3, we report that three out of four developers consider the number of stars before using or contributing to GitHub projects. This result indicates that projects that are already popular tend to be selected in place of the less popular ones. In the literature, this phenomenon is often referred to as the "rich get richer" [Wu et al., 2007; Gábor Kondor et al., 2014; Fang et al., 2014], meaning that the most popular repositories tend to attract more users and contributors than the less popular ones.

We also report that GitHub developers star repositories mainly to show appreciation (52.5%), to bookmark (51.1%), and because they used or are using the projects (36.7%). These results show that GitHub stars is a relevant metric that effectively groups different motivations, which can be summarized in a positive signal to the repository' maintainers.

In Chapter 3, we characterize the number of stars of GitHub projects. Regarding the selection of GitHub projects for empirical studies based on number of stars by empirical software engineering researchers, 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, right after the project public release.

In a survey with project owners, we reveal that the major reason for *moderate*, *fast*, and *viral* growth is active promotion. Regarding the selection of GitHub projects based on number of stars by researchers, the following observations are derived from our findings: (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).

Regarding the strategies for increasing the number of stars of GitHub repositories, active promotion is the most consistent one according to project owners. Indeed, in Chapter 4 we show that successful posts on Hacker News have a positive impact on the number of stars on the days following these posts. Moreover, we reveal that more that 61% of these posts relate to announcements of software releases. The cor-

relation between releases and the increase on the number of stars was also explored in Section 3.4, where we show that there is an acceleration in the number of stars gained after minor and major releases. In summary, promoting mainly releases on social news aggregator sites may increase the chances of GitHub projects receiving more stars.

6.4 Future Work

Future work includes 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 popularity 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) may provide additional insights on the factors that impact the classification of a project in the proposed growth patterns [Tantithamthavorn et al., 2017].

Regarding the channels used by developers to promote their projects, future work include an investigation of the content promoted by the projects. For example, in Chapter 4, we showed that successful Hacker News posts usually have a relevant impact on the popularity of GitHub projects. Thus, the understanding of the content and context of successful posts can guide developers on defining promotion strategies for their projects.

Regarding the popularity prediction, future work may include the investigation of prediction models that also consider the programming languages. In Chapter 5, we reported that models that consider the repository growth pattern improve the predictions and reduce the amount of data required to predict. Thus, the consideration of other characteristics of repositories may improve the predictions' accuracy. Finally, the investigation of different approaches to predict popularity, e.g., epidemic models [Wang et al., 2013] and machine learning models [Mohri et al., 2012], may improve our current results.

Bibliography

Aggarwal, K., Hindle, A., and Stroulia, E. (2014). Co-evolution of project documentation and popularity within GitHub. In *11th Working Conference on Mining Software Repositories (MSR)*, pages 360--363.

Ahmed, M., Spagna, S., Huici, F., and Niccolini, S. (2013). A peek into the future: predicting the evolution of popularity in user generated content. In *6th International Conference on Web Search and Data Mining (WSDM)*, pages 607--616.

Ali, M., Joorabchi, M. E., and Mesbah, A. (2017). Same app, different app stores: A comparative study. In *4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 79--90.

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

Archer, E. (2013). *rfPermute: Estimate Permutation p-Values for Random Forest Importance Metrics*.

Babaei, M., Kulshrestha, J., Chakraborty, A., Benevenuto, F., Gummadi, K. P., and Weller, A. (2018). Purple Feed: Identifying High Consensus News Posts on Social Media. In *AAAI/ACM Conference on Artifical Intelligence, Ethics & Society (AIES)*.

Bajic, D. and Lyons, K. (2011). Leveraging social media to gather user feedback for software development. In *2nd International Workshop on Web 2.0 for Software Engineering*, pages 1--6.

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

Bianco, V. D., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., and Tosi, D. (2012). A study on OSS marketing and communication strategies. In *8th International Conference on Open Source Systems (OSS)*, pages 338--343.

Bissyande, T. F., Thung, F., Lo, D., Jiang, L., and Reveillere, L. (2013). Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *37th Annual International Computer Software and Applications Conference (COMPSAC)*, pages 303--312.

Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. (2016). Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology*, 70:30 – 39.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In *32nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 1--11.

Borges, H., Valente, M. T., Hora, A. C., and Coelho, J. (2015). On the popularity of GitHub applications: A preliminary note. *CoRR*, abs/1507.00604.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5--32.

Capra, E., Francalanci, C., Merlo, F., and Rossi-Lamastra, C. (2011). Firms' involvement in open source projects: a trade-off between software structural quality and popularity. *Journal of Systems and Software*, 84(1):144–161.

Chatzopoulou, G., Sheng, C., and Faloutsos, M. (2010). A first step towards understanding popularity in YouTube. In *30th IEEE International Conference on Computer Communications Workshops (INFOCOM)*, pages 1--6.

Coelho, J. and Valente, M. T. (2017). Why modern open source projects fail. In *25th International Symposium on the Foundations of Software Engineering (FSE)*, pages 186--196.

Comino, S., Manenti, F. M., and Parisi, M. L. (2007). From planning to mature: On the success of open source projects. *Research Policy*, 36(10):1575--1586.

Corral, L. and Fronza, I. (2015). 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)*, pages 22–32.

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

Couto, C., Pires, P., Valente, M. T., Bigonha, R., and Anquetil, N. (2014). Predicting software defects with causality tests. *Journal of Systems and Software*, 93:24--41.

Cruzes, D. S. and Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. In *5th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 275--284.

Datta, D. and Kajanan, S. (2013). Do app launch times impact their subsequent commercial success? an analytical approach. In *International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, pages 205–210.

Fang, Y., Huang, H., Jian, P., Xin, X., and Feng, C. (2014). Self-adaptive topic model: A solution to the problem of "rich topics get richer". *China Communications*, 11(12):35–43.

Figueiredo, F. (2013). On the prediction of popularity of trends and hits for user generated videos. In *6th International Conference on Web Search and Data Mining (WSDM)*, pages 741--746.

Figueiredo, F., Almeida, J. M., Gonçalves, M. A., and Benevenuto, F. (2014). On the dynamics of social media popularity. *ACM Transactions on Internet Technology*, 14(4):1--23.

Figueiredo, F., Benevenuto, F., and Almeida, J. M. (2011). The tube over time: Characterizing popularity growth of YouTube videos. In *4th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 745--754.

Freedman, D. A. (2009). *Statistical Models: Theory and Practice*. Cambridge University Press.

Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. (2013). 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)*, pages 1276--1284.

Gousios, G., Pinzger, M., and van Deursen, A. (2014). An exploratory study of the pull-based software development model. In *36th International Conference on Software Engineering (ICSE)*, pages 345--355.

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

Guerrouj, L., Azad, S., and Rigby, P. C. (2015). The influence of app churn on app success and StackOverflow discussions. In *22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 321--330.

Guerrouj, L. and Baysal, O. (2016). Investigating the Android apps' success: an empirical study. In *24th International Conference on Program Comprehension (ICPC)*, pages 1--4.

Gábor Kondor, D., Pósfai, M., Csabai, I., and Vattay (2014). Do the rich get richer? an empirical analysis of the bitcoin transaction network. *PLOS ONE*, 9(2):1–10.

Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *Transactions on Software Engineering*, 38(6):1276--1304.

Hansson, C., Dittrich, Y., and Randall, D. (2006). How to include users in the development of off-the-shelf software: A case for complementing participatory design with agile development. In *39th Annual Hawaii International Conference on System Sciences (HICSS)*, pages 175c–175c.

Hartigan, J. A. (1975). *Clustering algorithms*. John Wiley & Sons, Inc.

Hinkle, D. E., Wiersma, W., and Jurs, S. G. (2003). *Applied Statistics for the Behavioral Sciences*. Houghton Mifflin.

Hora, A., Valente, M. T., Robbes, R., and Anquetil, N. (2016). When should internal interfaces be promoted to public? In *24th International Symposium on the Foundations of Software Engineering (FSE)*, pages 280--291.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., and Zhang, L. (2016). Why and how developers fork what from whom in GitHub. *Empirical Software Engineering*, 22(1):1--32.

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

Jiang, J., Zhang, L., and Li, L. (2013). Understanding project dissemination on a social coding site. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 132--141.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories (MSR)*, pages 92--101.

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

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

Lehmann, J., alves, G., Ramasco, J. J., and Cattuto, C. (2012). Dynamical classes of collective attention in Twitter. In *21st International Conference on World Wide Web (WWW)*, pages 251--260.

Linares-Vasquez, M., Bavota, G., Bernal-Cardenas, C., Penta, D. M., Oliveto, R., and Poshyvanyk, D. (2013). API change and fault proneness: a threat to the success of Android apps. In *9th Foundations of Software Engineering (FSE)*, pages 477--487.

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

Ma, Z., Sun, A., and Cong, G. (2012). Will this #hashtag be popular tomorrow? In *35th International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1173--1174.

Ma, Z., Sun, A., and Cong, G. (2013). On predicting the popularity of newly emerging hashtags in Twitter. *Journal of the American Society for Information Science and Technology*, 64(7):1399--1410.

Martin, W., Sarro, F., and Harman, M. (2016). Causal impact analysis for app releases in Google Play. In *24th International Symposium on the Foundations of Software Engineering (FSE)*, pages 1--12.

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

Menasce, D. A. and Almeida, V. (2001). *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

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

Oshiro, T. M., Perez, P. S., and Baranauskas, J. (2012). How many trees in a random forest? In *8th International Workshop on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pages 154--168.

Pagano, D. and Maalej, W. (2011). How do developers blog?: An exploratory study. In *8th Working Conference on Mining Software Repositories (MSR)*, pages 123--132.

Palomba, F., Vasquez, L.-M., Bavota, G., Oliveto, R., Penta, D. M., Poshyvanyk, D., and Lucia, D. A. (2015). User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *31st International Conference on Software Maintenance and Evolution (ICSME)*, pages 291–300.

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

Pinto, H., Almeida, J. M., and Alves, G. A. (2013). Using early view patterns to predict the popularity of YouTube videos. In *6th International Conference on Web Search and Data Mining (WSDM)*, pages 365–374.

Provost, F. and Fawcett, T. (2001). Robust classification for imprecise environments. *Machine learning*, 42(3):203--231.

Rakha, M. S., Shang, W., and Hassan, A. E. (2016). Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*, 21(5):1960--1989.

Roy, S. D., Mei, T., Zeng, W., and Li, S. (2013). Towards cross-domain learning for social video popularity prediction. *IEEE Transactions on Multimedia*, 15(6):1255--1267.

Sajnani, H., Saini, V., Ossher, J., and Lopes, C. V. (2014). Is popularity a measure of quality? an analysis of Maven components. In *30th Software Maintenance and Evolution (ICSME)*, pages 231–240.

Singer, L., Filho, F. F., and Storey, M.-A. (2014). Software engineering at the speed of light: how developers stay current using Twitter. In *36th International Conference on Software Engineering (ICSE)*, pages 211--221.

Szabo, G. and Huberman, B. A. (2010). Predicting the popularity of online content. *Communications of the ACM*, 53(8):80.

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

Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). 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)*, pages 1--10.

Tsay, J., Dabbish, L., and Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in GitHub. In *36th International Conference on Software Engineering (ICSE)*, pages 356--366.

Tsur, O. and Rappoport, A. (2012). What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In *5th International Conference on Web Search and Data Mining (WSDM)*, pages 643--652.

Vasilescu, B., Filkov, V., and Serebrenik, A. (2013). Stackoverflow and GitHub: Associations between software development and crowdsourced knowledge. In *6th International Conference on Social Computing (SocialCom)*, pages 188--195.

Vlas, R., Robinson, W., and Vlas, C. (2017). Evolutionary software requirements factors and their effect on open source project attractiveness. In *50th Hawaii International Conference on System Sciences (HICSS)*, pages 1--11.

Wang, H., Li, Y., Feng, Z., and Feng, L. (2013). Retweeting analysis and prediction in microblogs: An epidemic inspired approach. *China Communications*, 10(3):13–24.

Weber, S. and Luo, J. (2014). What makes an open source code popular on GitHub? In *13th IEEE International Conference on Data Mining Workshop (ICDW)*, pages 851--855.

Wolpert, D. H. and Macready, W. G. (1999). An efficient method to estimate bagging's generalization error. *Machine Learning*, 35(1):41--55.

Wu, M., Jiang, Q., and Zhang, Y. (2007). Worrisome rich-get-richer? not the true story! In *7th International Conference on Computer and Information Technology (CIT)*, pages 194–199.

Wu, Y., Kropczynski, J., Shih, P. C., and Carroll, J. M. (2014). Exploring the ecosystem of software developers on GitHub and other platforms. In *17th Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, pages 265--268.

Yang, J. and Leskovec, J. (2011). Patterns of temporal variation in online media. In *4th International Conference on Web Search and Data Mining (WSDM)*, pages 177--186.

Yates, R., Neto, B., et al. (1999). *Modern information retrieval*. ACM press New York.

Yu, Y., Wang, H., Filkov, V., Devanbu, P., and Vasilescu, B. (2015). Wait for it: determinants of pull request evaluation latency on GitHub. In *12th Working Conference on Mining Software Repositories (MSR)*, pages 367--371.

Zhou, S., Stănciulescu, Ş., Leßenich, O., Xiong, Y., Wąsowski, A., and Kästner, C. (2018). Identifying features in forks. In *40th International Conference on Software Engineering (ICSE)*, pages 1–12.

Zhu, J., Zhou, M., and Mockus, A. (2014). 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)*, pages 1--4.