Studying Android App Popularity by Cross-Linking GitHub and Google Play Store

John Businge,* Moses Openja,* David Kavaler,† Engineer Bainomugisha,‡ Foutse Khomh,§ and Vladimir Filkov†
*Mbarara University of Science and Technology, Mbarara, Uganda
†Makerere University, Kampala, Uganda
‡University of California, Davis, California, United States
§SWAT Lab., École Polytechnique de Montréal, Montréal,Canada

***Abstract*—The incredible success of the mobile App economy has been attracting software developers hoping for new or repeated success. Surviving in the fierce competitive App market involves in part planning ahead of time for the success of the App on the marketplace. Prior research has shown that App success can be viewed through its proxy–popularity. An important question, then, is *what factors differentiates popular from unpopular Apps?* GitHub, a software project forge, and Google Play store, an app market, are both crowdsourced, and provide some publicly available data that can be used to cross-link source code and app download popularity. In this study, we examined how technical and social features of Open Source Software Apps, mined from two crowdsourced websites, relate to App popularity. We observed that both the technical and the social factors play significant roles in explaining App popularity. However, the combined factors have a low effect size in explaining App popularity, as measured by *average user rating* on Google Play. Interestingly on GitHub, we found that *social factors have a higher power in explaining the popularity compared to all the technical factors we investigated*.**

***Index Terms*—Android Apps, GitHub, Google Play, Apps Popularity, Social factors, Statistical Modeling**

I. Introduction

In recent years, the mobile software development community has exploded with mobile applications (hereafter referred as apps) and has maintained an upward trajectory. As of the first quarter of 2018, in the leading app markets, Google Play and Apple App stores, users were able to choose from 2 million and 3.8 million apps, respectively.[1] This translates into a tremendous success for the app economy, e.g., in 2016, gaming apps generated 50.4 billion U.S dollars in revenue and projected to reach 105.2 billion U.S. dollars by 2021[1]. Unsurprisingly, Open Source developers are also flocking to these markets, hoping to find success. For example, as of September 2018, GitHub reported over 750K Open Source repositories that relate to Android, totaling 37K contributors[2]. Thus, for an Open Source app developer to survive in the fierce competitive market, it is important to plan ahead of time for the success of their apps on the marketplace. An important question then would be: *what factors differentiates successful from unsuccessful open source apps?*

Stewart and Ammeter [1] state that the popularity (i.e., extent to which an Open Source project attracts end user attention and developer effort/attention) is a potential indicator of the project's success. It can be argued that the planning should start during the development process, before an app is released on the marketplace, by minding the factors that the developers have control over that associate with or even affect app popularity. But which factors are appropriate to study as determinants of app popularity? A number of recent studies have investigated how app popularity relates to aspects of apps that developers can control [1]–[10]. Most of those studies investigated how technical aspects of the apps relate to App popularity.

Many apps on Google Play Store are Open Source, made on social coding platforms like GitHub, where the social interactions among developers and the ecosystem project environments likely play a role in the apps life. To add to the body of knowledge presented in previous studies, here we take a holistic approach and investigate a comprehensive collection of both technical and social measures, seeking to understand how they associate with Open Source app popularity.

Many app store apps have their code and development traces available on social coding websites. Studies have shown that social aspects in software development influence on code quality [11]–[13]. The quality of an app can have an impact on its success or failure. For example, if an app on GitHub has poor quality (i.e., hard to maintain code), it may have few contributors joining and, in the long run, it might be abandoned if the main contributor leaves. In our study, we combine social and technical factors mined from an App marketplace (Google Play store) and a developer social coding website (GitHub). We operationalize app popularity along quantifiable dimensions: user downloads and user average rating in Google Play as well as active forks on GitHub. The number of the App's active forks, the number of App user downloads, and user average ratings can be considered *proxies of success* [14], [15], which readily map onto our intuitive notions of popularity.

The main contributions of this paper are as follows:

1) We present a holistic approach of investigating App popularity by analyzing factors mined from two crowdsourced websites and combining both technical and social factors.
2) We show that technical App attributes, e.g., level of activity, are associated with App popularity on both GitHub and the Google play store.
3) We find that Apps with more minor contributors are also associated with increased popularity on both Google Play

[1]http://www.statista.com/statistics/276623/
[2]https://github.com/search?q=Android

and GitHub. We also show that apps with many major contributors are associated with lower popularity on both Google Play store and GitHub.

4) Relating to App's popularity on GitHub, we found that *social factors have a higher power in explaining popularity, compared to all the technical factors we investigated*. We discuss possible consequences of the social aspect metrics for practice.

In the rest of the paper, we discuss related work in Section II, the experimental setup in Section III-A, followed by the results and discussion, implications, threats to validity and the conclusion sections.

II. Related Work

App popularity measures the extent to which developers and end-users are involved in or use project artifacts. Meaningful measures of App popularity can be important to App managers or owners in assessing their artifact success. Several studies have investigated what makes mobile Apps popular on the marketplace; other studies have looked at popularity of projects on social coding websites. Below we give an overview of representatives of both.

A. Popularity of Apps on Google Play Store

Linares-Vásquez et al. [2] extracted data on Android APIs from Apps on Google Play and mined the entire change history from the APIs Git repositories. They discovered that change- and fault-proneness of these underlying APIs negatively impacted App success. Guerrouj et al. [3] extracted Android APIs used by Android Apps and then captured the addition and removal of those Android API elements between two consecutive releases of the App. The authors discovered that high App churn related to Android API methods used by the Apps was associated to poor user rating on the Google Play store. Lu et al. [5] extracted a taxonomy of 108 features from Apps APK files and others from the Wandoujia–a Chinese App store. The authors discovered a number of features that relate to popularity of the Apps in terms of downloads and user rating. Tian et al. [6] investigated factors extracted from the Apps' APK files and related factors from the Google play store. The authors discovered that the size of an App, and the target SDK version of an App, are the most influential factors associated with the App rating on Google Play store. Martin et al. record time-series information about popular Google Play apps and investigate how release frequency can affect an app's performance, as measured by rating, popularity and number of user reviews [16]. 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. Mojica et al. investigated the impact of integrating many many different ad libraries on an App's rating on Google Plat store. They found no evidence that the number of ad libraries in an app is related to its possible rating in the app store. However, they found that integrating certain ad libraries can negatively impact an App's rating.

B. App Popularity on Social Coding Websites

Weber and Luo attempt to differentiate popular and unpopular Python projects on GitHub using machine learning techniques [7]. They found that in-code features are more important than author metadata features. Zho et al. study the frequency of folders used by 140 thousands GitHub projects and the results suggest that the use of standard folders (e.g., doc, test, examples) may have an impact on project popularity [8]. Aggarwal et al. studied the effect of social interactions on GitHub projects' documentation [9]. They conclude that popular projects tend to attract more documentation collaborators. Borges et al. investigated the factors that influence popularity on GitHub repositories and found out that the main factors include: programming language, application domain and introduction of new features [10]. Stewart and Ammeter carried out an exploratory study of factors influencing the popularity of Open Source Project [1] using 240 open source projects registered on the freshmeat Website. They found that the more active a project is in terms of posting new releases and making announcements, the more attention it receives from the community. Borges and Valente conducted a survey with 791 developers and found out that developers star GitHub repositories due to three major reasons (which frequently overlap): to show appreciation to projects, to bookmark a project, and because they are using a project.

Each of the above studies have investigated the factors that relate App popularity by extracting their data from either Google Play store or social coding website. We observed that no study has tried to investigate App popularity by merging data from two crowdsourced websites. Furthermore, Constantinou and Mens [17] state that developing software is both a *social* and *technical* activity. *Social* activities involve the interaction, communication, and collaboration of multiple contributors to the same project and across interdependent projects. Most of the discussed studies mainly concentrate on investigating popularity mainly using the technical factors. We have not come across studies investigating popularity combing both technical and social factors. In our study, we take a holistic approach where we combine App factors mined both GitHub and Google Play store as well as assess how the combination of both technical and social factors relate to App popularity.

C. Popularity in other Social Networks

There are studies that report on popularity in social networks. Chatzopoulou et al. [18] analyze popularity of YouTube videos by looking at properties and patterns metrics, stating that several popularity metrics are highly correlated. Zongyang et al. [19] propose methods to predict the popularity of new hashtags on Twitter. Their main focus was to identify and evaluate the effectiveness of content and contextual features derived from tweets annotated with candidate hashtags. They demonstrated that contextual features were more effective than content features.

TABLE I: Collected metrics that may affect App popularity on Github and the Google Play store

Dimension	**Variable Name**	**Explanation**	**Rationale**
App popularity Github	TotalForks	Total number of accumulated forks until December 31, 2017	*Dependent Variable*
	ActiveForks	Active forks of an App (those that have at least one commit since the fork date)	
App popularity Google Play store	AverageRating	Average rating of the five level stars	*Dependent Variables*
	Downloads	User downloads on Play store	
	TotalStars	Total stars of the app	Used as a control variable in the model of AverageRating
Code Authorship Metrics (**Social aspects**)	majorDev	Number of major code authors	Open source projects' ability to attract and retain volunteer contributors is an important element of project success [15]. MVA tells us to what extent the App has shared ownership. Code authorship metrics have been shown to have an association with software quality in Android Apps [13]. Thus, we hypothesize that authorship metrics will have a related effect on App popularity.
	minorDev	Number of minor code authors	
	totalDev	Total number of code authors (Team size)	
	MVA	Most valuable author	
Contributor experience (**Social aspects**)	TotalReposMajDev	Total number of repositories the major contributors of the App have authored with at least one commit.	Related previous studies [20]–[25] have shown that contributor experience is an important factor for project maintenance. We hypothesize that Apps with experienced major developers have a high chance of being popular.
	TotalCommitsMajDev	Total number of commits the major contributors authored in the repositories of variable *TotalReposMajDev* above.	
	TotalChangedLOCMajDev	Total changed lines of code from commits authored by the major contributors across all repositories in which they are active	
Level of activity (**Technical aspects**)	TotalCommmits	Total number of commits in an App	More important than the sheer number of developers is their contribution to a project. The level of activity of contributors in submitting code may be useful as an indicator of project popularity [15].
	TotalChangedLOC	Total number of changed lines of code of in an App	
	Tags	Total number of Tags in an App (named commits)	
	PullRequests	Closed pull requests	
	Issues	Closes Issues	
App size (**Technical aspects**)	SLOC	The size of the master branch of an App on Github measured in source lines of code	This variable is included as a control for levels of activity
App Time (**Technical aspects**)	FDateGH	The date of the second commit of an App on Github	FDateGH and LDateGH tell us the time frame of project activity of Github, while GPDate tells us how recent updates on Google Play were made.
	LDateGH	The last commit date of an App on Github	
	GPDate	Update date of an App on Google Play	
	Weeks	The number of weeks from the App's 2nd commit to the last commit or December 31, 2017 (the last day of collecting App statistics on Github)	Indication of how long an App has been actively maintained on Github. We use this as a control in our models.
	Inactivity	The time interval in months between the App's last commit and December 31, 2017 (the last day of collecting App statistics on Github)	More inactivity may indicate a lack of lasting interest in the project.

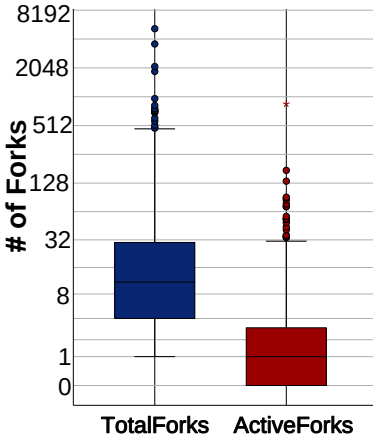

Fig. 1: Boxplots for TotalForks and ActiveForks. The sample size used in the figure is our final data set of 814 Apps used in our analysis (Section III-B)

.

III. Study Design

In this paper, we set out to empirically investigate the factors that relate to App popularity on both Google Play and GitHub. In this section, we introduce our research questions, discuss the variables used in the study and present our data extraction approach. Furthermore, we describe our model construction and model analysis approaches.

In Section II we presented a number of studies that have investigated App popularity. In them, measures of popularity on the social coding website GitHub include number of stars and number of forks. The study of Borges et al. [10], which is most directly related to this one, found the number of stars and forks to be very highly correlated. Thus, we consider `ActiveForks`[3] as the measure of popularity on GitHub. Next, we present our research questions.

> On a social coding websites like GitHub, new developers do not have to develop their repos from scratch; if there is a related repo, they can fork the mainline repo for free, make updates to it, and thereby start a new project. While making updates, they may discover bugs or new features that they wish to contribute back to the mainline repo through pull requests or reporting an issue and hence improve on the mainline repo's quality. A mainline repo that is popular (i.e., forked many times) indicates that developers beyond the mainline likely use its code, and hence may increase its success through continued development [26]. A mainline repo having fewer forks may suggest that it has fewer developers maintaining it; that could result in neglect or abandonment if the main contributor(s) leaves the project. Knowing which features of a project associate with more forks, and which with fewer, can possibly help developers understand the reasons for a project thriving versus withering. Thus, we wish to investigate what factors are related to App popularity, i.e., number of forks, on GitHub.

1) *RQ1. What are the factors that relate to the App's number of active forks on GitHub?*
   In the App marketplace, if an App is popular (i.e., has many downloads or a high user rating), the users can report bugs and suggest new features to the developers to improve the App. Popularity of an App on the marketplace can translate into possibly attracting more developers and to higher monetary success, as Apps generate revenue in a number of different ways, including: charging users small amounts of money, in-app purchases, and ads. Apps that are not popular are likelier to generate smaller returns on investments and less likely to attract developers. As in RQ1, we want to investigate factors that developers can possibly control relating to app popularity.
2) *RQ2. What are the factors that relate to the App's number of downloads and user star rating on Google Play?*

By knowing factors that correlate with popularity, app owners/managers/developers can monitor them and act if the recorded metrics go in a direction that can be detrimental to the popularity of the app.

A. Variables used in the study

In this section we discuss the different variables used in our models. Table I lists each dimension, which variables fall into these dimensions, a summary explanation, along with grounded rationale behind our choices. We further describe each dimension and the dependent and independent variables as follows:

Dependent Variables *App Popularity*: aspects that measure how popular an App is, *e.g.*, the number of App downloads and average user ratings reported on the Google Play store, total number of GitHub forks, and number of *active forks* on GitHub.

Independent variables

In Table I column–`Rationale` we discuss reasons why we chose these specific independent variables below to estimate the App popularity.

Contributor experience–Social factors: aspects that measure the experience of the App developer, *e.g.*, how many repositories has the developer contributed to on GitHub? and what is the size of these contributions?

Level of activity–technical factors: aspects that measure contributions to the App, *e.g.*, number commits, number of tagged commits, and number changed lines of code.

App size–technical factors: aspects that measure an App's size, *e.g.* source lines of code.

App Contributors–social factors: aspects that represent human effort in an App's development, such as App team

[3]During our data collection, we found that some Apps had many total forks. However, the number of active forks (i.e., those with at least a single commit since the fork date) are few. As can be seen from Figure 1 there is a significant difference in the distribution of `TotalForks` and `ActiveForks`. The median for `TotalForks` is around 11 while the median for `ActiveForks` is one. This implies that most forks of an App are not active after the fork date. Similarly for repository stars, it is possible for one to star a repo when they do not really mean it. However, unlike for forks, it is not possible to discover active stars.

size, number of major developers, number of minor developers, and most valuable author (MVA).

Although a developer may not directly control the metrics in *Level of activity* dimension, they can promote their app in developers community if they want to boost these metrics. With these carefully considered metrics and motivational background in place, we are interested in two measures of popularity: user App downloads and average rating on Google Play store and active forks on GitHub.

This leads us to our research questions:

B. Data Collection and Metric Extraction

The data used in the study was extracted from GitHub and the Google Play store. We mined data from GitHub using the GitHub Rest API v3[4] and data from Google play using BeautifulSoup[5]. The method of linking between GitHub and Google Play, along with metric extraction, is described as follows:

1) Using the GitHub's Rest API v3 we identified a total of 9,610 repositories with the following criteria: 1) contain the word "Android app" in the repo name, description or `README.md` file; 2) are not forks; 3) contain at least two forks; 4) are written in any programming language; and 5) were created no later than 31-12-2017. We used the criteria of having at least two forks to reduce the chance of finding student assignments, which could pollute our results [27].
2) To ensure that the repositories we identified are indeed real Android apps, we searched for an `AndroidManifest.xml` file in each repository and, if it existed, we identified the package name indicated in the file. We then looked for the App manually on Google Play using the extracted package name. This step filtered out more repositories from our list, since some repositories may have no manifest file or might have no corresponding app on Google Play. We obtained 1,103 repositories representing an actual Google Play app.
3) Next, we manually looked at the list of the 1,103 apps to identify any apps sharing the package name, which would mean they are linked to the same app on Google Play. We removed 45 apps that have duplicate package names, leaving 1,058 apps. We speculate that duplicate package names are a result of repositories cloning other apps' source code and including it in their own.
4) From the remaining 1,058, we eliminated apps with less than six commits in their lifetime, according to the median number of commits in GitHub projects found by prior work [28]. After this preprocesing step, we were left with 919 apps in our data set.
5) For the remaining 919 repos, we collected GitHub commits. A previous study related to ours [13] observed that the size of first commit (*i.e.*, changed lines of code) for some Android projects was very large compared to later commits. The authors reported that the large size of the first commit could be as a result of projects that began development elsewhere and later joined GitHub, transferring their entire prior project in a single commit. To this end, in collecting the commits, we start from the second commit. For each commit, we collected meta-data that included: author name, e-mail address, login name, and changed lines of code.
6) Related studies [13], [26] observed that some contributors of the applications used more than one account to make their commits, which causes them appear as different contributors. We employ the heuristics used by the previous authors of name merging to remove possible duplicate contributor information. We merged the details of two contributors into one using the following heuristics, in the order mentioned: 1) if they possess the same *login ID*, 2) possess the same *full names*, and 3) possess the same *e-mail* prefix (*i.e.*, prior to the email domain name). Using the mentioned heuristics, of the 919 apps, we merged 145 contributors who participated in 144 apps. It is possible that two distinct individuals could have the same full name in a project. However, since the apps are very small with few contributors, the possibility of having more duplicate developers is very likely compared to having two distinct individuals.
7) For each of the 919 apps in the dataset, we labeled each of the contributors in an App as a major or minor developer based on their percentage of contributions (*cf.* Section III-D for a detailed explanation of how we label the major and minor contributors). One of the metrics to extract is experience of the major developers in an App (*i.e.*, the total contributions of the major developers across all repositories they commit to on GitHub). Mining the repositories a developer has contributed to requires one to have a developer's GitHub login name. For this reason, we eliminated all apps that had at least one major developer with no login name; we do the same elimination across minor developers. Out of the 919 apps, 101 apps were deleted, leaving 814 apps in our data set.
8) For each of the 814 apps left in our dataset, we clone the master branch from GitHub. We then counted the source lines of code (`SLOC`—control variable) using the the tool *cloc*[6].
9) Finally, from our final dataset of 814 apps, we wrote a Python script using BeautifulSoup[7] that crawls the Google Play store page for each App to extract store-related features (`GPDate`, `Downloads`, `TotalStars`, `AverageRating`, and `App category`).

C. Directly Extracted Metrics

Below we present the remaining extracted metrics listed in TableI that do not require specialized computations, calculated per App.

[4]https://developer.GitHub.com/v3/

[5]https://www.crummy.com/software/BeautifulSoup/

[6]https://GitHub.com/AlDanial/cloc

[7]https://www.crummy.com/software/BeautifulSoup/

`Weeks`: the number of weeks since the second commit until the last commit or until 31-12-2017.

`PullRequests`: the number of closed pull requests that were up to 31-12-2017.

`Issues`: the number of closed issues that were created up to 31-12-2017.

`TotalForks`: the total number of forks created up to 31-12-2017.

`ActiveForks`: the number of forks (subset of `TotalForks`) that contain at least one commit after the fork date up to 31-12-2017.

`TotalReposMajDev`: For each of the `majorDev` (defined below), the sum of all repos that are listed in the developers' profile where the developer has made at least one commit, up to 31-12-2017.

`TotalCommitsMajDev`: For each repo in `TotalReposMajDev`, the sum of all commits authored by the major developers up to 31-12-2017.

`TotalChangedLOCMajDev`: For each commit in `TotalCommitsMajDev`, the sum of changed lines of code (*i.e.*, the total changed lines of code across all major developers' repos on GitHub).

D. Computed Metrics

To define major (`majorDev`) and minor (`minorDev`) developers, we employ the methodology described in [11], [13], [29]. We computed code authorship metric values following the definition proposed by Bird *et al.* [11] and used by Rahman and Devanbu [29], which consists of calculating the proportion of contribution for each author. If C^b lines are changed on a repository r in a time interval t, and there are a total number of m distinct authors, and the number of lines contributed by author a in the time interval t is C^b_t, then the contribution ratio of a is $r^C_a = \frac{C^b_t}{C^b}$.

We label the major (`majorDev`) and minor (`minorDev`) developers in an App using the threshold implemented by Businge et al. [13]. Here we briefly recapitulate the implementation method. We sort the developer contribution ratios in descending order and thereafter we sum them up as illustrated in Equation 1. $sum_{0.8}$ is the summation of the ratios starting with the largest ratio r^C_1 (highest ratio) to r^C_n, where r^C_n is the first ratio where $sum_{0.8} \geq 0.8$ and $n \leq m$.

$$sum_{0.8} = \sum_{i=1}^{n} r_i \tag{1}$$

Using Equation 1, we label major authors if they have been identified within $sum_{0.8}$, and minor authors in the remaining $sum_{0.2}$; *i.e.*, the threshold to separate the developers is 80%.

To compute the `MVA` (most valuable author) metric, we consider the author with the highest ratio. For example, if an application has two code authors having contribution ratios of 0.93 and 0.07, the `MVA` value will be 0.93. However, we note that there are cases of low `MVA` metric for some projects.

E. Multiple Linear Regression Models and Tuning

In this section, we discuss how we build multiple linear regression models to gain insight towards answering our RQs. As studied by similar works [11], [13], [30]–[32], our main goal for building App popularity models is not to *predict* App popularity, but to *understand* the relationship between our explanatory variables of interest and App popularity, measured by number of forks and user downloads on the Google Play store. In the regression models with dependent variables `ActiveForks`, `Downloads` and `AverageRating`, along with explanatory variables (predictors), one can observe which variables have an effect on dependent variables, how large the effect is, in what direction, and, *e.g.*, how much of the variance in the number of downloads is explained by our metrics. In our models a number of control variables that include: `SLOC` was used to control the effect of different App sizes, `Weeks`, `FDateGH` and `LDateGH` were used to control the effect of different apps first commit dates and last commit dates on GitHub, respectively, and `LDateGB` was used to control the effect of different Google Play update dates.

Before building the models, we made careful preparations in order to have model results that can be trusted, and thus useful for inference. First, we standardized the variables by subtracting the the mean and dividing by the standard deviation. This allowed our variables to be on the same relative scale (which is important since we are building models across different applications). We also utilized a *log* transform on most explanatory variables to stabilize the variance and improve model fit, when appropriate [33].

Second, we performed feature selection by first considering which features may be theoretically important, guided by rationale listed in Table I, and removing those which introduce high levels of multicollinearity. Highly correlated explanatory variables can cause issues for inference in a given model, as multicollinearity can negatively affect standard error estimation (*i.e.*, incorrect p-values), and thus should be removed. We achieved this by consulting correlation heatmaps and considering the Variance Inflation Factor (VIF) [34] using `R`. All variables in the final models had VIFs of under 5, as guided by standard rule of thumb [33], [35].

Third, because the interpretation of the models' results can be influenced by the presence of redundant variables, we checked for redundant variables using the `redun` function in the `rms R` package [36]. However, we found that none of the explanatory variables that survived our correlation analysis were redundant.

Fourth, for the ordinary least squares (OLS) [34] regression to be reliably interpreted, one must inspect the residuals of fitted models. Non-normality of the residuals is often attributable to the skewness of the variables. In our data set, variables that are found to be skewed are log transformed to stabilize the variance and improve the model fit, whenever appropriate [35]. Finally, we took special care to make sure that the most important modeling assumptions of OLS regression were met, namely: 1) homoscedasticity [34] (by examining *residual vs. fitted* plots), 2) linear independence (mentioned above, by removing highly

TABLE II: Category counts for Apps studied in models

Category	# Apps	% Apps	Category	# Apps	% Apps
Tools	204	25.1	Video Players	15	1.8
Productivity	87	10.7	Personalization	15	1.8
Education	74	9.1	Lifestyle	15	1.8
Libraries and Demo	46	5.7	Games	13	1.6
Social	41	5.0	Photography	10	1.2
Communication	33	4.1	Shopping	10	1.2
Entertainment	33	4.1	Weather	10	1.2
Music and Audio	31	3.8	Sports	7	0.9
News and Magazines	25	3.1	Events	5	0.6
Maps and Navigation	25	3.1	Medical	4	0.5
Books and Reference	24	2.9	Comics	2	0.2
Business	22	2.7	House and Home	2	0.2
Travel and Local	20	2.5	Art and Design	2	0.2
Health and Fitness	20	2.5	Food and Drink	1	0.1
Finance	17	2.1	Auto and Vehicles	1	0.1

TABLE III: Descriptive statistics of factors

Factor	Mean	Min	1st Quartile	Median	3rd Quartile	Max
SLOC	16.6K	38	24.4K	5.2K	13K	831.3K
Weeks	101.7	2	34	78	142	446
Inactivity	30.4	0	7	20	44	195
PullRequests	29.6	0	0	2	12	33.3K
Issues	74.4	0	1	8	37.8	66.7K
TotalForks	59.0	1	4	11	29.8	5.2K
ActiveForks	5.3	0	0	1	3	862
Tags	10.4	0	0	2	10	267
TotalCommits	319.5	6	37	104.5	285.3	26.3K
TotalChangedLOC	224K	66	7.6K	25.8K	89.6K	13,514K
majorDev	1.3	1	1	1	1	9
minorDev	4.2	0	0	1	4	111
Totaldev	5.5	1	1	2	5	116
MVA	88.3	15.2	82.05	99.3	100	100
TotalReposMajDev	5.8	1	2	4	7	64
TotalCommitsMajDev	545.5	4	89	236.5	614.25	22K
TotalChangedLOCMajDev	562K	242	21K	58K	173870.3	23,227K
AverageRating	3.8	0	3.7	4.2	4.6	5
Downloads	13,346K	0	500	5K	50K	10,000,000K

correlated variables according to VIF), and 3) normality of errors (by examining *normal qq plots*). In addition, we also consult the *Cook's distance vs. leverage plots* to identify any potentially overly influential outliers to examine for validity. This resulted in the removal of six points in our data set which improved the model fit (based on R^2 value) while having minimal effect on the estimated model coefficients (*i.e.,* no variable coefficients changed signs or significance).

IV. Study Results

In this section, we present and discuss the results of our findings for RQ1 & RQ2.

A. Descriptive Statistics

The first step in our analysis consisted of examining various descriptive statistics of the variables described earlier. The apps are written in 18 programming languages with the majority being written in Java (716), Kotlin (37) and JavaScript (29).

Tables II & III present descriptive statistics of the variables used in our models as well as other variables describing the studied apps. One can draw a number of insights from these descriptive statistics. Looking at Table II, we see that the studied apps represent the wide variety of App categories available on Google Play. In Table III, when comparing the values of `SLOC`, `TotalCommits` and `TotalChangedLOC`, we see an indication that the applications have gone through multiple releases throughout their lifetime, which reduces the probability that they are student class assignments rather than real products. Looking at the values of `Weeks` and `Inactivity`, we also observe that the applications have been in existence for a number of months. Lastly, looking at the column values of Min, 1st Quartile, Median, 3rd Quartile and Max, we observe that most of the variables have a heavily right skewed distribution.

TABLE IV: Linear Model coefficients and the ANOVA for ActiveForks

Linear Model					**Analysis of Variance**			
Coefficients:	**Estimate**	t-value	p-value		**Sum Sq**	F-value	p-value	
(Intercept)	55.060	1.951	0.051	.				
SLOC	0.017	0.76	0.448		58.9	207.3	0.000	***
FDateGH	0.000	-1.899	0.058	.	16.0	56.2	0.000	***
TotalChangedLOC	-0.016	-0.751	0.453		23.0	80.8	0.000	***
majorDev	-0.231	-2.945	0.003	**	65.8	231.7	0.000	***
minorDev	0.288	5.613	0.000	***	329.5	1160.4	0.000	***
TotalReposMajDev	0.181	4.15	0.000	***	7.7	27.1	0.000	***
TotalCommitsMajDev	-0.033	-1.391	0.165		1.3	4.7	0.030	*
TotalChangedLOCMajDev	-0.022	-1.103	0.270		0.6	2.1	0.146	
AverageRating	-0.009	-0.59	0.555		0.1	0.2	0.674	
Downloads	0.019	2.371	0.018	*	2.5	8.9	0.003	**
Issues	0.142	8.817	0.000	***	24.8	87.4	0.000	***
Tags	-0.023	-1.42	0.156		0.5	1.9	0.173	
LDateGH	0.000	2.035	0.042	*	0.4	1.4	0.232	
GPDate	0.000	-1.657	0.098	.	0.6	2.0	0.154	
MinDev^2	0.119	7.805	0.000	***	17.3	60.9	0.000	***
Residuals					221.5			

Significance codes: 0.000 (***), 0.001 (**), 0.01 (*), 0.05 (.), >= 0.1 ()
Multiple R-squared: 0.7125, Adjusted R-squared: 0.707
Social factors highlighted

B. OLS Model Results

RQ1. What factors contribute to App popularity on GitHub?

In Table IV we present the results of the OLS regression model for the dependent variable *ActiveForks*. Explanatory variables are the coefficients presented in the first column of the table. The model yields a multiple R-squared of 0.7125 and an adjusted R-squared of 0.707. This tells us that the model captures 71.3% of the variance of the dependent variable. The high value of 71.3% is not surprising since the both the dependent and most of the independent variables are extracted from one source–GitHub and possible that some variables could be related.

Looking at the t-values, p-values, and *Sum Sq* in Table IV, we see that the most influential covariates are *Code authorship metrics—social factors (highlighted)*—`minorDev` (***) (*Sum Sq* 329.5) and `majorDev` (**) (*Sum Sq* 65.8). Though other covariates such as the *technical factors* of `TotalReposMajDev`, `Issues`, `Downloads`, and `LDateGH` show significant p-values in the regression, their contribution in explaining the variance of the dependent variable is minimal when we consider the *Sum Sq* in the ANOVA analysis. To this end, we shall focus our discussion on `minorDev` and `majorDev`.

RQ1–Discussion of Findings

Looking at the coefficient estimate of `minorDev`–social factor, the positive sign indicates that apps on GitHub that have high values of minor authors do have high values of active forks. This finding is not surprising as it can be believed that an App is owned by a small number of authors (*i.e.*, contributors with write access), while the majority of the other authors contribute through the fork and pull request model. In the fork and pull request model contributors fork a project, make updates to the project privately (*i.e.*, to an active fork), and later propagate the changes to the main-line via pull request.

Looking at the value of `majorDev`, the negative sign indicates that apps on GitHub with a high number of major

TABLE V: Linear Model coefficients and the ANOVA for Downloads and AverageRating

	Downloads						AverageRating					
	Linear Model			Analysis of Variance			Linear Model			Analysis of Variance		
Coefficients:	**Estimate**	p-value		**Sum Sq**	p-value		**Estimate**	p-value		**Sum Sq**	p-value	
(Intercept)	1.33E+03	0	***				6.88E-01	0.861				
SLOC	2.59E-01	0.007	**	215	0	***	1.03E-01	0.031	*	20.45	0	***
ActiveFork	2.88E-01	0.024	*	322	0	***	-1.30E-02	0.833		0.02	0.901	
TotalChangedLOC	-4.00E-01	0.00	***	27	0.024	*	4.40E-02	0.335		0.84	0.427	
Tags	6.35E-02	0.343		108	0	***	1.30E-01	0.001	**	14.92	0	***
majorDev	-8.75E-01	0.01	*	61	0.001	***	2.80E-02	0.892		1.87	0.237	
minorDev	3.59E-01	0.017	*	50	0.002	**	-1.11E-01	0.139		4.52	0.066	.
TotalReposMajDev	-5.86E-01	0.002	**	0	0.806		1.38E-01	0.147		0.05	0.853	
TotalCommitsMajDev	2.84E-01	0.006	**	98	0	***	-2.10E-02	0.676		7.93	0.015	*
TotalChangedLOCMajDev	2.83E-01	0.001	**	44	0.004	**	-1.35E-01	0.001	**	16.58	0	***
Inactivity	7.56E-02	0.326		239	0	***	-6.50E-02	0.059	.	1.8	0.246	
PullRequests	-2.10E-02	0.794		49	0.002	**	-6.80E-02	0.152		5.48	0.043	*
FDateGH	-6.64E-05	0	***	959	0	***	4.64E-02	0.756		1.02	0.325	
LDateGH	-1.10E-03	0	***	3	0.457		0.00E+00	0.01	**	2.69	0.157	
GPDate	1.35E-03	0	***	173	0	***	0.00E+00	0.026	*	6.67	0.026	*
TotalStars							2.26E-01	0	***	186.67	0	***
Residuals				4176						1051.45		

Significance codes: 0.000 (***), 0.001 (**), 0.01 (*), 0.05 (.), >= 0.1 ()
Downloads: Multiple R-squared: 0.3599, Adjusted R-squared: 0.3485
AverageRating: Multiple R-squared: 0.2046, Adjusted R-squared: 0.1905
Social factors highlighted

authors have a low number of active forks. A clue to the possible reasons behind this observed behavior can be related to the work of Bird *et al.* [11]—who studied the relationship between bugs in software modules and code authorship – and Businge *et al.* [13]—who studied the relationship between bugs in small-sized Android applications and code authorship. Both authors found that modules/applications with many major contributors (*i.e.*, more shared authorship; lower values of `MVA`) are associated with more bugs. The reason behind this may be related to the theory of "too many cooks spoil the broth" [11], suggesting that as the number of developers increase, coordination in development efforts becomes too complex to accomplish. Because of the complexity in coordination, we posit that a possible resulting problem could be that submitted pull requests and/or issues could take more time to resolve. This may demoralize new contributors interested in forking and providing changes to these apps; hence, fewer active forks are realized.

To validate our stated conjectures, we carried out the following deeper investigation. From our data set of 814 apps, we carefully selected projects using the following criteria: 1) at least one `PullRequests` and at least one `Issues`; 2) at least two total contributors (*i.e.*, `TotalDev`$>= 2$)

This filtered dataset consisted of 469 apps. For each `PullRequests`/`Issues` in each of the apps, we computed the number of days between `PullRequests`/`Issues` creation time and closing time. We then computed the total number of days it took to close all the `PullRequests`/`Issues` in each of the apps (*days-to-close-PullRequests* and *days-to-close-Issues*, as well as the average number of days to closure (*mean-days-to-close-PullRequests* and *mean-days-to-close-Issues*).

In our experiment, we wanted to ascertain the relationship between *mean-days-to-close-PullRequests*/*mean-days-to-close-Issues* and `MVA` (`MVA` metric shows us the level of code authorship). We divided the values of `MVA` into two groups whereby values of `MVA` $>= 80\%$ were labeled *less-shared-authorship* and those that were $< 80\%$ labeled *more-shared-authorship*. The choice of the threshold 80% is based on the methodology discussed in Section III-D. We found that variable *mean-days-to-close-PullRequests* had a mean of 13.6 average days for the *less-shared-authorship* group and a mean of 27.6 average days for the *more-shared-authorship* group. This tells us that the average days to close a pull request in the *less-shared-authorship* group is less than the average days to close a pull request in the *more-shared-authorship* group. To ascertain if there was a statistically significant difference between the groups, we then run a Mann-Whitney's U test and calculate the effect size of Mann-Whitney's U test to evaluate the difference between the groups [37]. We found a significant effect on the groups with p-value < 0.003 and effect size (r) = 0.14 (*small size effect*). We did not find the *Mann-Whitney U test statistic* for *mean-days-to-close-Issues* statistically significant.

From the results and the findings of RQ1, it is worth to note that the social factors of `minorDev` and `majorDev` have *more significant contributions while predicting App popularity on GitHub compared to the rest of the technical factors*.

RQ2. What factors contribute to an App's popularity on Google Play?

In Table V we present the results of the OLS regression models for the dependent variables of `Downloads` and `AverageRating`. Explanatory variables are presented in the first column of the table, with estimated coefficients in the second. The model yields a multiple R-squared of 0.3599 and an adjusted R-squared of 0.3485 for `Downloads` and R-squared of 0.2046 and an adjusted R-squared of 0.1905 for `AverageRating`. This tells us that the proportion of the variance in the dependent variable that is predictable from the independent variable(s) is 36.0% and 20.5% for `Downloads` and `AverageRating`, respectively. The low values of R-squared are not surprising since the independent varibales and the dependent variables are extracted from different repositories—GitHub and Google Play, respectively.

As shown, for the `Downloads` model, variables representing *level of activity* (`TotalChangedLOC`***), *App Popularity* (`ActiveFork`***), *Contributor experience* (`TotalReposMajDev`***, `TotalCommitsMajDev`**,

TotalChangedLOCMajDev***), App size (SLOC*), *App Time* (FDateGH***, LDateGH**, GPDate***) and *Code authorship* (majorDev* minorDev*) are the most significant predictors of the dependent variable. Looking at the ANOVA in Table V, Sum Sq (column five), we observe significant contributors of the value of the dependent variable (in order of importance) to be FDateGH, ActiveForks, Inactivity, SLOC, GPDate, Tags, TotalCommitsMajDev, majorDev, minorDev, PullRequests, TotalChangedLOCMj and TotalChangedLOC.

For the AverageRating model, variables representing *level of activity* (Tags***), *App size* (SLOC*), *Contributor experience* (TotalChangedLOCMajDev**), *App Time* (LDateGH** and GPDate*) are the most significant predictors of the dependent variable. Looking at the ANOVA in Table V, Sum Sq (column 11), we observe significant contributors of the value of the dependent variable (in order of importance) to be TotalChangedLOCMajDev***, SLOC***, Tags***, TotalStars*** TotalCommitsMajDev*

RQ2–Discussion of Findings

In the results presented for RQ2 above, we note some interesting observations. We discuss the variables that have both coefficients with significant t-value as well as **Sum Sq**> 20. For AverageRating, we observe that apart from TotalStars– used as a control variable, the rest of the variables have low values of **Sum Sq**. Below we discuss variables of interest relating to App popularity on Google Play.

Technical factors

ActiveForks: The positive coefficient of this variable tells us that apps that are popular on GitHub, are also popular on Google Play. A possible explanation for this could be that popular apps on GitHub have contributors who improve the quality and/or introduce interesting features that continuously attract new users on Google Play.

GPDate: The positive coefficient of this variable tells us that apps newer updates on Google Play have more user downloads. This could mean that users on Google Play prefer continuously updated apps, perhaps due to fear that apps which are not updated often may be obsolete.

TotalChangedLOC: The negative coefficient of this variable tells us that apps on GitHub with more churn have low values of user downloads on Google Play. Our finding could be related to the findings of Nagappan and Ball [38], who found that an increase in relative code churn measures is accompanied by an increase in system defect density. This could potentially mean that an App with high churn is accompanied with post release faults that may cause downloaded apps on users phones to fail or crash. This would result in these users giving an App a bad rating and writing a bad review, resulting in the App failing to attract new users.

Social factors

TotalCommitsMajDev, TotalChangedLOCMajDev: The positive coefficient of these variables tell us that apps on GitHub with experienced major contributors have more user downloads on Google Play. This could mean that experienced major contributors produce apps of high quality and/or introduce interesting features; hence, the apps attract new users on Google Play.

majorDev: The negative coefficient of this variable tells us that apps on GitHub with many major contributors (*i.e.*, shared ownership) have low values of user downloads on Google Play. The study of Businge *et al.* [13] discovered that apps with shared authorship (*i.e.*, many major developers) were found to be more fault-prone. This could imply that when users download apps with higher levels of shared authorship on GitHub, they run into issues with said apps, and report these issues on Google Play. Eventually, new users may read these reviews, and choose not to download the App as a result.

minorDev: The positive coefficient of this variable tells us that apps on GitHub with a high number of minor authors have high values of user downloads on Google Play. This could mean that minor authors contribute quality code and/or introduce interesting features; hence, the App continuously attracts new users on Google Play.

Like in RQ1, we also decided to carry out further investigation to verify the conjecture that apps with higher levels of shared authorship (i.e., more major developers) are more buggy. From our dataset of 814 apps, we carefully selected projects using the following criteria: At least two total contributors (*i.e.*, totalDev>= 8). The filtered dataset contained a total of 119 apps. We thereafter used the code quality analysis tool SonarQube[8] to extract bug reports from the source code of the 119 apps. Of the 119 apps, SonarQube could not parse source nine apps for some reason we could not establish. Unlike in RQ1, in RQ2 we considered a few apps in our experiment due to the semi-automated nature of SonarQube in parsing the source code of the apps.

In our experiment, we wanted to ascertain the relationship between the number of bugs and MVA (a metric that shows us the level of shared code authorship). Like in RQ1, We divided the values of MVA into two groups whereby values of MVA$>=$ 80% were labeled *less-shared-authorship* and those that were $< 80\%$ labeled *more-shared-authorship*. We found that the variable "number of bugs" had a mean of 13.02 for the group with MVA $>= 80\%$, and a mean of 25.75 for the MVA $< 80\%$ group, i.e., the average in the *less-shared-authorship* group is less than the average bugs in the *more-shared-authorship* group. Like in RQ1, to ascertain if there was a statistically significant difference between the groups, we then run a Mann-Whitney's U test and calculate the effect size of Mann-Whitney's U test to evaluate the difference in the groups [37]. *We found a significant effect of on the groups with p-value* < 0.027 *and effect size (r)* $= 0.21$ *(small size effect).*

For the comparison of the social factors and the technical factors in explaining App popularity on Google Play, we note that although some of the technical factors are more important, the social factors also do have a significant contribution.

[8]https://www.sonarqube.org/

V. Implications of Overall RQ1 & RQ2 Discussion

We note that in addition to the technical factors that have previously been extensively studied, we have observed that social factors do have a significant contribution in explaining app popularity on both Google Play and GitHub. We make the following recommendations in relation to the social factors of `majorDev` and `minorDev`.

`majorDev`: We have discovered that apps on GitHub with many major contributors are less popular on both Google Play store and GitHub. The low popularity of Google Play store could be attributed to the fact that we found that apps with many major developers are associated with more bugs than those with fewer major developers. The low popularity on GitHub could be attributed to the fact that we found apps with many major developers have a slow response time in resolving pull requests. To this end, we recommend that Android applications with lower levels of code authorship **should be reviewed with more scrutiny** for both possible bugs and longer pull request resolution response time. Android application project teams on GitHub could make use of the `MVA` metric to scrutinize applications with low values of `MVA`.

`minorDev`: We have also discovered that having more minor contributors of an app is associated with the App's popularity on both GitHub and Google Play store, with a compounding effect as the number increases (indicated by the significant positive squared term). We encourage apps teams on GitHub to accept and encourage contributions from the community in order for their App to become popular on both GitHub and Google Play store.

Discussion–In Comparison with Related Work

In Section II we discussed a number of studies that have investigated App popularity. All the studies performed their investigation using factors from only one crowdsourced website (i.e., either GitHub or Google Play). Furthermore, the studies mainly concentrated on App technical factors that could be related to popularity. In this study, we took a holistic approach where we combined factors from the two crowdsourced websites to investigate App popularity as well as combining both technical and social factors. We have observed that both the technical and social factors do play a significant role in explaining App popularity on the two crowdsourced websites. We observe that the combined factors from GitHub and Google Play have a low effect in explaining App popularity measured by *average user rating* on Google Play. Interestingly on GitHub, we have discovered that *social factors have a higher power in explaining the popularity compared to all the technical factors we investigated*. The data used in our study can be found online[9].

VI. Threats to Validity and Verifiability

Though we have sought to make sure that all of our data was gathered and linked correctly and that our models are statistically robust, we note some potential threats to validity.

[9]https://sites.google.com/view/app-saner-2019

First, as mentioned in Section III-B in the method of linking GitHub and Google Play–Step 3, we performed a manual inspection of the data. It is possible that we could have missed a few cases that skipped our eyes unnoticed. To a lesser extent, this could threaten our findings.

Second, one may question whether or not we have sufficient number of projects (observations) in our data set compared to the total number of apps on the Google Play store and GitHub. As mentioned in Section III-A, we carefully selected the apps used in our dataset, removing apps that could be considered experimental or student projects. Thus, our dataset can be considered representative for the notions of popularity we have modeled.

Third, in collecting our popularity data of the apps on Google Play we use the latest number of downloads and total number of reviews since Google Play does not keep history of the app data. It is possible that an app can have large swings in popularity following a code change. That may threaten the results of our findings.

Lastly, there is an *internal validity* threat related to the tool (i.e., SonarQube) used to extract bug information, and to compute size and complexity metrics. SonarQube is well maintained and extensively used by practitioners. However, as most static analysis tools, it is not 100% precise. It is possible that some of the bugs considered in this study will never be experienced either by developers or app users.

VII. Conclusion and Future Work

In this study, we examined how the App's combined factors of technical and social factors, mined from two crowdsourced websites, relate to App popularity. We have observed that both the technical and social factors do play a significant role in explaining App popularity on the two crowdsourced websites. We observe that the combined factors have a low effect in explaining App popularity measured by *average user rating* on Google Play. Interestingly on GitHub, we have discovered that *social factors have a higher power in explaining the popularity compared to all the technical factors we investigated*.

For future studies, fusing heterogenous data sets, as we did, can apparently allow for comprehensive study of factors involved in popularity of apps. Naturally, one wanders if adding more social web data, e.g., info on developers communication about the projects, or prior links between them, can lead to even higher resolution of determinant factors. Furthermore, in the study we looked at only two crowdsourced websites where extracted info. This work can be extended by looking at other crowdsourced websites for example Travis-CI and Stackoverflow which are rich sources of project data.

Acknowledgement

We would like to acknowledgement the efforts of Prof. Prem Devambu and Prof. Bogdan Vascilescu who helped us in the early stages of the research. This work was supported by the Sida/BRIGHT project under the Makerere-Sweden bilateral research programme 2015-2020.

REFERENCES

[1] K. Stewart and T. Ammeter, "An exploratory study of factors influencing the level of vitality and popularity of open source projects," in *Proceedings of the Twenty-Third International Conference on Information Systems*, 2002, pp. 14–17.

[2] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: A threat to the success of android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 477–487. [Online]. Available: http://doi.acm.org/10.1145/2491411.2491428

[3] L. Guerrouj, S. Azad, and P. C. Rigby, "The influence of app churn on app success and stackoverflow discussions," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 321–330.

[4] G. Bavota, M. Linares-Vásquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "The impact of api change- and fault-proneness on the user ratings of android apps," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 384–407, April 2015.

[5] X. Lu, Z. Chen, X. Liu, H. Li, T. Xie, and Q. Mei, "Prado: Predicting app adoption by learning the correlation between developer-controllable properties and user behaviors," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 79:1–79:30, Sep. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130944

[6] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, Sept 2015, pp. 301–310.

[7] S. Weber and J. Luo, "What makes an open source code popular on git hub?" in *2014 IEEE International Conference on Data Mining Workshop*, Dec 2014, pp. 851–855.

[8] J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: A case study of github repositories," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 30:1–30:4. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652564

[9] K. Aggarwal, A. Hindle, and E. Stroulia, "Co-evolution of project documentation and popularity within github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 360–363. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597120

[10] H. Borges, A. Hora, and M. T. Valente, "Predicting the popularity of github repositories," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE 2016. New York, NY, USA: ACM, 2016, pp. 9:1–9:10. [Online]. Available: http://doi.acm.org/10.1145/2972958.2972966

[11] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[12] M. Foucault, C. Teyton, D. Lo, X. Blanc, and J.-R. Falleri, "On the usefulness of ownership metrics in open-source software projects," *Inf. Softw. Tech.*, vol. 64, no. C, pp. 102–112, Aug. 2015.

[13] J. Businge, S. Kawuma, E. Bainomugisha, F. Khomh, and E. Nabaasa, "Code authorship and fault-proneness of open-source android applications: An empirical study," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE. New York, NY, USA: ACM, 2017, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/3127005.3127009

[14] W. H. DeLone and E. R. McLean, "Information systems success: the quest for the dependent variable," *Information systems research*, vol. 3, no. 1, p. 60, 1992.

[15] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: Theory and measures," *Software Process–Improvement and Practice*, vol. 11, no. 2, p. 123–148, 2006.

[16] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in google play," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 435–446. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950320

[17] E. Constantinou and T. Mens, "Socio-technical evolution of the ruby ecosystem in github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 34–44.

[18] G. Chatzopoulou, C. Sheng, and M. Faloutsos, "A first step towards understanding popularity in youtube," in *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, March 2010, pp. 1–6.

[19] Z. Ma, A. Sun, and G. Cong, "On predicting the popularity of newly emerging hashtags in twitter," *Journal of the American Society for Information Science and Technology*, vol. 64, no. 7, pp. 1399–1410. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.22844

[20] J. Businge, A. Serebrenik, and M. G. J. van den Brand, "An empirical study of the evolution of Eclipse third-party plug-ins," in *EVOL-IWPSE'10*. ACM, 2010, pp. 63–72.

[21] J. Businge, A. Serebrenik, and M. G. J. v. Brand, "Analyzing the Eclipse API usage: Putting the developer in the loop," in *17th European Conference on Software Maintenance and Reengineering (CSMR'13)*, 2013, pp. 37–46.

[22] J. Businge, M. van den, and A. Serebrenik, "Eclipse API usage: The good and the bad," *Software Quality Journal*, vol. 23, pp. 107–141, 2013.

[23] J. Businge, A. Serebrenik, and M. van den Brand, "Compatibility prediction of eclipse third-party plug-ins in new eclipse releases," in *12th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2012, Riva del Garda, Italy, September 23-24, 2012*, 2012, pp. 164–173.

[24] J. Businge, S. Kawuma, E. Bainomugisha, and A. Serebrenik, "How stable are eclipse application framework internal interfaces?" in *SANER*, 2019.

[25] J. Businge, A. Serebrenik, and M. G. J. van den Brand, "Survival of Eclipse third-party plug-ins," in *ICSM*, 2012, pp. 368–377.

[26] J. Businge, M. Openja, S. Nadi, E. Bainomugisha, and T. Berger, "Clone-based variability management in the android ecosystem," in *Proceedings of the 34th International Conference on Software Maintenance and Evolution*, ser. ICSME, 2018.

[27] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, Dec 2017.

[28] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 92–101. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597074

[29] F. Rahman and P. Devanbu, "Ownership, experience and defects: A fine-grained study of authorship," in *Pro. of the 33rd International Conference on Software Engineering*, 2011.

[30] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Pro. of the 38th International Conference on Software Engineering*, 2016.

[31] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[32] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering*, 2012.

[33] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[34] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, fifth edition ed. Chapman and Hall/CRC, 2011.

[35] C. J. Nachtsheim, J. Neter, M. H. Kutner, and W. Wasserman, "Applied linear regression models," *McGraw-Hill Irwin*, 2004.

[36] F. E. Harrell Jr., *Regression Modeling Strategies*, 2015.

[37] A. Field, *Discovering statistics using SPSS*, 3rd ed. Sage, 2009.

[38] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 284–292. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062514