The Role of Data Filtering in Open Source Software Ranking and Selection

Addi Malviya-Thakur
University of Tennessee
Oak Ridge National Laboratory
USA
amalviya@vols.utk.edu

Audris Mockus
University of Tennessee, Knoxville
Vilnius University
USA and Lithuania
audris@utk.edu

ABSTRACT

Faced with over 100M open source projects, a more manageable small subset is needed for most empirical investigations. Over half of the research papers in leading venues investigated filtering projects by some measure of popularity with explicit or implicit arguments that unpopular projects are not of interest, may not even represent "real" software projects, or that less popular projects are not worthy of study. However, such filtering may have enormous effects on the results of the studies if and precisely because the sought-out response or prediction is in any way related to the filtering criteria.

This paper exemplifies the impact of this common practice on research outcomes, specifically how filtering of software projects on GitHub based on inherent characteristics affects the assessment of their popularity. Using a dataset of over 100,000 repositories, we used multiple regression to model the number of stars –a commonly used proxy for popularity– based on factors such as the number of commits, the duration of the project, the number of authors and the number of core developers. Our control model included the entire dataset, while a second filtered model considered only projects with ten or more authors. The results indicated that while certain characteristics of the repository consistently predict popularity, the filtering process significantly alters the relationships between these characteristics and the response. We found that the number of commits exhibited a positive correlation with popularity in the control sample but showed a negative correlation in the filtered sample. These findings highlight the potential biases introduced by data filtering and emphasize the need for careful sample selection in empirical research of mining software repositories. We recommend that empirical work should either analyze complete datasets such as World of Code, or employ stratified random sampling from a complete dataset to ensure that filtering is not biasing the results.

CCS CONCEPTS

• **Software and its engineering**; • **Human-centered computing → Empirical studies in collaborative and social computing**;

KEYWORDS

Empirical software engineering, Missing data problem, Software engineering research, Filtering, Sampling, Mining software repositories

ACM Reference Format:
Addi Malviya-Thakur and Audris Mockus. 2024. The Role of Data Filtering in Open Source Software Ranking and Selection. In *Proceedings of International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

WSESE 2024, 2024, Lisbon, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Volumes of public data that could be extracted from open source projects that use version control and issue tracking systems have been widely exploited by research that seeks to better understand software development projects. Such studies increasingly use a large collection of projects to achieve some generality of the results. As it is extremely difficult to collect these data from all OSS projects, most research focuses on selecting a manageable subset of projects from the largest "forge", GitHub. Once projects are selected according to some criteria, data of these projects are extracted, cleaned, and analyzed to identify a relationship between some predictors and outcomes investigated in specific research [8, 12, 26]. In fact, in more than half of the cases of such analysis obtained from leading conferences, the projects were selected based on some measure of popularity. Another study [7], investigated methods used in 93 publications that analyzed the mining of GitHub repositories with respect to reported methods, datasets, and limitations and found numerous issues with the dataset collection process and size and poor sampling techniques. The authors of [4] filtered the top 2500 public repositories on GitHub with high stars. The goal of this restriction was to concentrate on the peculiarities of the widely used GitHub systems and the difficulties that come with human examination. The "star" ratings are a static representation of the community's preferences at a certain moment in time (namely, the past) rather than current choices. It also does not show which packages are used the most or which are growing or losing popularity. In [13, 25], the authors proposed ranking the code and filtering the repositories based on the entity of the code and its usage. This approach is based on ~12,000 Java projects filtered by the presence of source code, resulting in the database consisting of 4600 repositories. Such filtering can surely exclude projects that are widely used and safeguards against phishing, intentional code spoofing, among others. Along the same line [1], the authors performed a filtering based on the static snapshot of code repositories based on one-time usage,

resulting in more than 200K filtered code bases. In [22], the authors illustrated the relationship between popularity and security level based on the top 20 widely used Java libraries. The filtering was based on usage rather than on more elaborate and finer-grained experiments. Research has shown that in open source software projects, factors like the number of branches, open issues, and contributors significantly influence popularity, with each factor having a large effect size [9]. Another study indicates that the number of followers and aspects such as programming language and commits by others at the time of newcomer joining are crucial for long-term contributor retention, highlighting these as top features in various time intervals [3]. Several research studies have highlighted the use of collaborative filtering as a way to identify key repositories to examine a multitude of scientific problems [5, 16, 24]. Such studies clearly raise the problem of trust, reproducibility, and validation resulting from incomplete data samples through heuristic filtering of open source repositories.

Authors in [6] identify the use of small data sets, poor sampling techniques and hard to replicate methodologies, while authors of [19] recognize that contemporary filtering methods may be insufficient means of identifying "real" software projects and trie to use machine learning models that rely on various repository statistics to identify "real" software projects. In particular, while our proposed work focuses on the methodological robustness of sampling strategies over time, [19] offers a practical tool for filtering engineered projects in large software repositories. Thus, [19] and our proposed work address limitations in current research practices but from different angles: one through methodological improvement and the other through a technical solution.

In summary, some measure of popularity or project activity is often used to filter out projects that are below certain thresholds for that measure or for prediction probability (in the case of ML models). As a result, many software projects are excluded from the subsequent analysis based on filtering criteria, resulting in the so-called "missing data".

Missing data, if not handled properly, distorts statistical results and degrades the prediction of machine learning methods. Filtering removes the response and all independent variables, while traditional missing data techniques are typically used when at least one of the predictors is present and uses imputation and/or weighting techniques to fill in missing values [14, 17]. The three types of missing data [14] are MCAR or "missing completely at random", MAR or "Missing at random," and MNAR or "missing not at random." Filtering would result in MCAR if the distribution of the response is independent of the filtering criteria, a quite unlikely scenario in most cases. The advantage of MCAR is that the analysis on the filtered data would be valid. Fortunately, the analysis would be valid even in MAR cases, which require that the missingness be conditionally independent given the observed variables, i.e., the fact that a value is missing depends only on the observed values. Since filtering excludes all values, subsequent analysis is equivalent to the listwise-deletion technique (i.e., when data rows with any missing values are excluded). It yields correct results only if the filtering criteria are completely unrelated to the response: an extremely unlikely scenario.

For example, if we predict the number of stars, and the likelihood that the number of stars is missing depends only on the number of authors, we need to include the number of authors in a regression model and impute the missing values for the number of stars to get valid results. Filtering, unfortunately, excludes entire data rows where the number of authors is low, hence any subsequent analysis would be invalid. MCAR assumption would not apply, because the probability that a value is missing depends on the number of authors. However, if we do not have the number of authors (in case of filtering) or simply do not include it in our estimation model, then even the MAR assumption is not satisfied. Such case is referred to as data not missing at random (MNAR). The MNAR data can be made to satisfy the MAR assumption if variables that characterize situations when a value is missing are added. Therefore, it is important to add variables that might predict the missing value mechanism to the dataset. With filtered data that is not possible.

In this study, we illustrate the often overlooked issue of missing data arising when software projects are filtered out of research datasets. We demonstrate the significant impact this can have on research outcomes, particularly through a case study that models the popularity of software projects.

Our recommended approach to solving this problem involves either using the full data when possible (research infrastructure such as World of Code [15] and archival collections such as Software Heritage [23] make complete datasets much easier to access) or the use of a stratified sampling technique [20] from the above-mentioned resources to effectively reduces the risk of bias, ensuring that the empirical findings remain representative despite excluded (missing) data. Our proposed use of stratified sampling might help addresses the concerns highlighted in [2], which emphasizes the rarity of representative sampling in software engineering research, proposing a solution that contributes to resolving the field's generalizability crisis. The key idea of the stratified sampling is that different populations (strata) of projects may have different properties based on certain criteria, for example, the mechanism that gets a single-author project many stars may be different from one for projects with thousands of contributors. Instead of getting data on all projects, a random sample is selected for each strata and then the results are extrapolated to the entire population. A simple random sampling would also work, but it would require a much larger sample size for the same variance of the desired estimates, e.g., a very large random sample is needed to include at least some of the very large (and extremely rare) projects.

First, we explained how filtering is related to missing data and how analysis on filtered data is equivalent to listwise deletion technique. Second, we provide an actual example based on a very large real dataset that demonstrates that filtering may completely change the conclusions. Third, we illustrate how stratified sampling could be applied to manage the scale of the analysis yet yield representative results.

We draw data from a collection of open source version control repositories that attempts to approximate the complete universe of open source software, World of Code (WoC) [15], in order to be able to draw a truly random sample and compare it with filtering approaches commonly used in empirical software engineering research.

The remainder of the paper introduces an illustrative example in Section 2, discusses limitations in Section 3, discusses stratified sampling in Section 3, and concludes in Section 6.

All data and code used in this study are publicly available in the replication package links: https://zenodo.org/records/10516681 and https://github.com/woc-hack/Assignments/

2 ILLUSTRATIVE EXAMPLE

This section illustrates the problem of changes in the rating resulting from missing and filtering the repositories based on certain attributes. To model the rating of software repositories and measure what causes the ratings to change, we rely on key inherent characteristics of the repositories and model these quantities to understand what causes some projects to have more stars than others.

2.1 Problem statement

We would like to explain the popularity of a GitHub repository by various other attributes. Our primary hypothesis is that repositories with more activity, more authors, and those that have been around for a longer time will be more popular. To operationalize these concepts, we use the number of stars as a measure of popularity, the number of commits as a measure of activity, the time since the repository was created as duration, and the number of individuals who committed code to the repository as the number of authors.

For simplicity, we used linear regression to model the number of stars, and the following sections describe precise ways in which measures were obtained and the sample was filtered.

Data Extraction. Since its inception, Free/Libre and open-source software (FLOSS) has dramatically impacted the software community and ecosystem. FLOSS is free software and open source software, so anyone can freely use, copy, study, and change the software in any way. World of code (WoC) has enabled research on the global properties of FLOSS [15]. The WoC collection of software repositories contains authors, projects, commits, blobs, dependencies, and a history of the FLOSS ecosystems. These data are updated regularly and contain billions of git objects. We will use WoC's extensive and frequently updated version control commit data collection of the entire FLOSS ecosystems for this study. WoC APIs are used to fetch and process data related to FLOSS repositories. We extract several attributes of repositories' to develop the ranking/popularity rating mechanism. To baseline the analysis and develop a control dataset, we have extracted more than 100K repositories containing the following attributes about each repository.

- project id (*p*)
- the starting date (*fr*)
- project number of stars (*ns*)
- the project duration (*dur*)
- number of authors (*na*)
- number of commits (*nc*)
- number of Core Developers (*nCore*)
- number of commits by Top Developers (*nc1*)

Please note that the quantities obtained are highly skewed; hence we do a log transform to make it more suitable for linear regression. Next, we begin by performing a correlation analysis on the extracted data to estimate the relationship among the attributes of the repositories. Understanding the correlations among the predictors informs the interpretation of the results of the model. To simplify the interpretation, we typically want to leave only one of the highly correlated predictors in the model [18].

Table 1: Correlation analysis based on collected data and analysis results using Spearman correlation method.

	ns	dur	na	nc	nCore	nc1
ns	1.00	0.26	0.16	0.17	0.08	0.15
dur	0.26	1.00	0.14	0.25	0.06	0.16
na	0.16	0.14	1.00	0.28	**0.70**	0.06
nc	0.17	0.25	0.28	1.00	0.16	**0.89**
nCore	0.08	0.06	**0.70**	0.16	1.00	-0.10
nc1	0.15	0.16	0.06	**0.89**	-0.10	1.00

Correlation Analysis. We begin with a correlation analysis of the independent variables extracted for the repositories to measure the strength of the linear relationship. This allows us to determine how much one variable changes due to the change in the other. A high correlation indicates a strong association between the two variables, whereas a low correlation indicates a poor relationship between the variables. We have used Spearman's correlation for this study. The results of the analysis shown in Table 1 illustrate a strong correlation between the number of authors and number of core developers, and the second between the number of commits and the number of commits by the top developers, This conclusion would help us to understand the interdependence of random variables in a large data set of the repository. Next, we define our approach to calculate the popularity as a function of the repository's set of independent variables.

In this section, we demonstrate how filtering can alter the calculation of the popularity rating of open source repositories. The meaning and calculation of the popularity vary based on the underlying formulation and the definition of the popularity itself. For example, some studies define popularity as the download of software among the practitioner community. In comparison, some use developer activities as a function of assigning popularity to open source repositories [25]. Others have used HITS-based influence analysis on graphs that represent the star relationship between Github users and repositories [10], among others [4, 11, 22]. However, whatever definition entails, we demonstrate how a chosen well-defined popularity rating of software can change when a varying number of repositories are studied.

Popularity rating. The software repository rating is measured as a linear combination of log of duration (*ldur*), number of commits (*lnc*), number of authors (*lna*), and number of core developers (*lncore*). We begin by defining a plausible rating definition as a combination of several repository attributes. Then we show whether such a set of attributes is a reliable mechanism to estimate the ratings. We propose the use of these attributes to estimate the ratings/rankings of repositories. However, in doing so, we also demonstrate the argument that repository ratings can vary as we modify the definition and the contribution of repository attributes in calculating the ratings. That said, let us assume we define multiple linear regression as follows,

Table 2: MLR with random control sample of 100K repositories

Attribute	Estimate	Std. error	t-value	Pr(>|t|)
ldur	0.155	0.0046	33.600	<2e-16
lnc	0.023	0.0070	3.100	0.0019
lna	0.795	0.0240	32.800	<2e-16
lnCore	-0.564	0.0350	-16.126	<2e-16

$$lm(lns \sim ldur + lnc + lna + lCore, data = za)$$

To determine the repository popularity rating, we employ multiple regression, is a statistical technique for predicting the outcome of a response variable by combining several explanatory variables. The linear relationship between explanatory (independent) and response (dependent) variables is represented using multiple linear regression.

Multiple regression is essentially an extension of ordinary least-squares regression because it uses more than one explanatory variable. We utilize this method to calculate the popularity rating using a linear combination of the attributes mentioned above. In addition, the relationship between many independent criteria and the calculation of the dependent popularity rating is investigated.

After each of the independent factors has been determined to predict the dependent attribute, the information on the numerous attributes can be used to provide an accurate prognosis on the level of effect they have on the outcome of the popularity rating. Next, we begin with an unfiltered random sample of repositories that serves as a control (baseline) for the repository ranking. Then, we show how filtered out or missing repositories from this control sample can alter the rating calculation as the estimates of the independent variables change. This illustrates that changes in filtering result in a change in the outcomes of the analysis.

Control Sample. We begin with the creation of a control sample of 100k repositories to serve as the baseline estimate for the independent variables contributing to the ranking of the repository. The control data will remain unchanged throughout the experiment. It will be used as a reference and for comparison with other filters that induce missing data and evaluate potential changes in the dependent variables. We test whether the independent variables (predictors) that include log of duration (*ldur*), log of number of commits (*lnc*), log of number of authors (*lna*), and log of number of core developers (*lncore*) go from positive (or reverse) on the dependent variable (project number of stars (*ns*)).

We show the results of the multiple regression analysis of control data in Table 2. The commits made by the developers (*lnc*) has an estimated value of 0.0232. Next, we examine the effect of filtering by the number of authors (*lna*).

Projects with 10+ authors. To illustrate by how much the filtering may alter the results, we select a sample by filtering the list of repositories containing at least 10+ authors or more. Since the number of stars is our response variable, any filtering by the number of stars would immediately bias the results. Instead, we filter by one of the predictors: the number of authors. Since the number of authors is included as a predictor in the model, it should not affect the modeled relationship between the number of authors and the number of stars, but filtering is equivalent to listwise-deletion and for the analysis to be valid an MCAR assumption needs to hold (and it does not).

Table 3: MLR results for more than ten authors

Attribute	Estimate	Std. error	t-value	Pr(>|t|)
ldur	0.57	0.08	6.98	2.01e-11 ***
lnc	-0.52	0.11	-4.55	7.93e-06 ***
lna	3.55	0.38	9.4	<2e-16 ***
lnCore	-2.74	0.32	-8.66	3.32e-16 ***

Next, we estimate the popularity ratings through the independent variables ($lm(lns \sim ldur + lnc + lna + lCore)$). The results of this are shown in Table 3. These results show that the number of commits has a negative influence when we filter the repositories by the number of authors. Note that the estimated value of log of the number of commits (*lnc*) in the control dataset results is positive, while the estimated value of *lnc* in this case for filtered repositories with 10+ authors became negative. This dichotomy in results further underscores the crucial point that even when the criteria for data exclusion, such as instances of missing data, are integrated as predictive variables in the analysis, the process of filtering can substantially alter the outcomes of empirical studies.

2.2 Proposed solution

We propose a practical solution for the classification and selection of open source software using stratified sampling. This approach involves dividing software projects into distinct groups before sampling, ensuring that various project types are represented. It addresses some issues like biases from unmeasured factors and missing data, common in software analyses. By weighting each group according to its prevalence in the real world, stratified sampling requires a much smaller sample size than a simple random sampling yet allows inference on the overall population. Next section discusses this in more details.

3 STRATIFIED SAMPLING

The field of polling has developed sophisticated sampling techniques that minimize the effort needed to obtain a reasonably accurate estimate of the outcome of, for example, an election. Since systems such as WoC make it easy to collect data for a range of properties for nearly all software projects, sampling is not technically needed. But research may still require an effort-intensive calculation of additional predictors or a manual qualitative analysis. As such, the need for sampling is almost always present. A simple random sample of even 1000 GitHub repositories would result in tiny, short-duration, inactive projects. Putting extra effort into obtaining additional measures for such a sample would not be advisable. However, depending on the hypothesis and likely relationships in the data, we may develop sampling procedures that minimize missing-data bias. In all cases, we should not filter by the value of the response variable, whether it be popularity, activity, or team size. Such filtering would almost certainly introduce a missing not at random situation if we do not measure and

include in the model *all* variables that affect the outcome and either include all rows (no filtering) or do a simple random sampling. However, it should be possible to filter by predictors that are included in the model assuming that there are no unmeasured predictors. Unfortunately, commonly used models in software engineering explain a relatively small fraction of the variability in the response, suggesting that many important predictors are not measured. The relationships may also vary according to the size of the predictors, further exacerbating biases introduced by filtering. As an approach to avoid the problem of simple random sampling, it is advisable to carefully design the sampling so that projects with different values of the characteristics that may affect the outcome are included. If we want to obtain results for the entire population, we should weight each strata according to its prevalence in the original population. The effects can also be investigated and compared among the different strata and help to obtain more easily and meaningfully generalizable results.

For example, in our case we may want to sample an equal number of projects n from the sub-populations of projects with one to k authors. Suppose that the population size at each k is N_k. Then we can fit a weighted multiple regression where the weights for the observations representing k-th strata would be $\frac{N_k}{n}$. Generally speaking, regression analysis and other inference methods are somewhat different for stratified sampling, see, e.g., [21] and the details are beyond the scope of this illustration.

Another important question is how to conduct a random or stratified sampling in the first place. First, existing research mostly ignores projects not on GitHub, including many important and large open source projects. Second, studies are restricted on sampling using GitHub APIs or mostly obsolete and partial public datasets and have to devise filtering criteria based on whats available in such datasets. Fortunately, several efforts such as Software Heritage [23] and World of Code [15] spent significant amount of effort to collect nearly complete collection of open source projects from all public forges. In fact, World of Code research infrastructure provides interested researchers with access and training and, among other features, includes extensive support for random or stratified sampling such as numerous summaries of software projects, authors, and even programming APIs.

4 DISCUSSION

This work focuses on demonstrating the impact of filtering on the results obtained in a simple case of linear or logistic regression. The specific example was chosen to be as simple as possible to avoid many other aspects of statistical analysis that are important for obtaining valid conclusions. We tried to illustrate only the most simple aspects of sampling and did not attempt to cover more sophisticated and largely unsolved problems, for example, sampling from large graphs or linking multiple data sources [18]. By selecting features that do not overlap with the filtering criteria, our aim is to mitigate the risk of distributional bias. In the future, we will explore the identification of a set of characteristics that are demonstrably independent of our filtering criterion for further analysis. The regression in an example is not an attempt to show any causal connection. All variables may conceivably affect each other. Therefore, it is not meaningful to argue whether the number of authors increases the popularity rating or whether the reverse may be true. Instead, the model simply shows that the variables are related in the samples considered. The performance of regression models can generally be assessed by looking at the significance of the model coefficients, the general value of the model R^2, and other diagnostics such as residuals and fit graphs. It is also important to note that some of the repositories may not even represent software projects but may be used for other purposes, so it may be important to exclude such irrelevant cases. However, the retention of legitimate software projects should be done in a way that avoids the biases described here. This may be done as described in Section 3.

5 LESSONS LEARNED

In the course of our demonstration, we've learned valuable lessons that shed light on the intricacies of data filtering and its implications for research in open-source software ranking and selection:

Impact of Data Filtering: This study highlights the substantial influence of data filtering on research outcomes. The act of filtering data based on specific criteria can have a profound impact on the relationships between project characteristics and the target variable. This often leads to results that diverge significantly from those obtained when using unfiltered data.

The Challenge of Missing Data: Another challenge emerges in the form of "missing data." Filtering projects based on certain criteria can result in incomplete datasets, introducing potential biases into the analysis. Filtering corresponds to analysis with missing data when all rows with any missing (filtered) value are excluded. We argue that the analysis of filtered data is valid only when the filtering criteria are independent of the response, a condition that is typically not satisfied.

Biases in Data Filtering: It is crucial to emphasize that data filtering should not be contingent on the value of the response variable (e.g., popularity). Instead, it should be guided by predictors included in the analysis model, aimed at circumventing the introduction of biases.

The Role of Stratified Sampling: To address issues arising from data filtering and missing data, we recommend the adoption of stratified sampling techniques. Stratified sampling involves categorizing software projects into distinct groups before sampling, ensuring equitable representation across various project types. This approach enhances the precision and applicability of the research findings.

Prudent Sample Selection: Researchers are advised to exercise caution when selecting samples for empirical studies involving software repositories. Thoughtful sample selection plays a pivotal role in mitigating potential biases introduced by data filtering.

Ensuring Research Reliability: This study underscores the paramount importance of guaranteeing the reliability of research results. Researchers are encouraged to perform diagnostic tests, use cross-validation, and consider longitudinal analysis to validate model assumptions and affirm the robustness and consistency of their findings over time.

6 CONCLUSION

Sampling strategies play a crucial role in data discovery science, as the findings are drawn from the data captured for analysis. In

this paper, we illustrate how convenience sampling or filtering by various criteria could affect the investigated relationship, leading to very different results, sometimes opposite to what would be obtained from the unfiltered data. These evaluations help to understand the robustness of the research results and ensure that the conclusions drawn from such studies are reliable and representative of the real world scenarios they intend to model. In the future, we plan on conducting a longitudinal analysis to see if the results hold over time. This is particularly important if the ratings change over time, as it could affect the relevance and applicability of the research findings.

ACKNOWLEDGEMENTS

The work was partially supported by National Science Foundation awards 1633437, 1901102, 1925615, and 22120429.

This manuscript has been authored by UT-Battelle, LLC, USA under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid up, irrevocable, worldwide license to publish or reproduce the published form of the manuscript, or allow others to do so, for U.S. Government purposes. The DOE will provide public access to these results in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

REFERENCES

[1] Sushil K. Bajracharya, Joel Ossher, and Cristina V. Lopes. 2010. Leveraging Usage Similarity for Effective Retrieval of Examples in Code Repositories. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (Santa Fe, New Mexico, USA) (FSE '10). Association for Computing Machinery, New York, NY, USA, 157–166. https://doi.org/10.1145/1882291.1882316

[2] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: a critical review and guidelines. Empirical Software Engineering 27 (2022), 94. Issue 4. https://doi.org/10.1007/s10664-021-10072-8

[3] Lingfeng Bao, Xin Xia, David Lo, and Gail C. Murphy. 2021. A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects. IEEE Transactions on Software Engineering 47, 6 (2021), 1277–1298. https://doi.org/10.1109/TSE.2019.2918536

[4] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). 334–344. https://doi.org/10.1109/ICSME.2016.31

[5] Liang Chen, Angyu Zheng, Yinglan Feng, Fenfang Xie, and Zibin Zheng. 2018. Software Service Recommendation Base on Collaborative Filtering Neural Network Model. In Service-Oriented Computing, Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu (Eds.). Springer International Publishing, Cham, 388–403.

[6] Valerio Cosentino, Javier L. Cánovas Izquierdo, and Jordi Cabot. 2017. A Systematic Mapping Study of Software Development With GitHub. IEEE Access 5 (2017), 7173–7192. https://doi.org/10.1109/ACCESS.2017.2682323

[7] Valerio Cosentino, Javier Luis, and Jordi Cabot. 2016. Findings from GitHub: Methods, Datasets and Limitations. In Proceedings of the 13th International Conference on Mining Software Repositories (Austin, Texas) (MSR '16). Association for Computing Machinery, 137–141. https://doi.org/10.1145/2901739.2901776

[8] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). https://doi.org/10.1109/MSR52588.2021.00074

[9] Junxiao Han, Shuiguang Deng, Xin Xia, Dongjing Wang, and Jianwei Yin. 2019. Characterization and Prediction of Popular Projects on GitHub. In 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 1. 21–26. https://doi.org/10.1109/COMPSAC.2019.00013

[10] Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. 2016. Influence analysis of Github repositories. SpringerPlus 1 (2016). https://doi.org/10.1186/s40064-016-2897-7

[11] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. 2005. Ranking significance of software components based on use relations. IEEE Transactions on Software Engineering (2005). https://doi.org/10.1109/TSE.2005.38

[12] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (Hyderabad, India) (MSR 2014). Association for Computing Machinery, New York, NY, USA, 92–101. https://doi.org/10.1145/2597073.2597074

[13] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. Data Mining and Knowledge Discovery 18, 2 (2009), 300–336. https://doi.org/10.1007/s10618-008-0118-x

[14] Roderick JA Little and Donald B Rubin. 2019. Statistical analysis with missing data. Vol. 793. John Wiley & Sons.

[15] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretzki, and Audris Mockus. 2020. World of Code: Enabling a Research Workflow for Mining and Analyzing the Universe of Open Source VCS data. International Journal of Empirical Software Engineering (2020). papers/WoC_EMSE.pdf

[16] Frank McCarey, Mel O Cinneide, and Nicholas Kushmerick. 2006. A Recommender Agent for Software Libraries: An Evaluation of Memory-Based and Model-Based Collaborative Filtering. In 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. 154–162. https://doi.org/10.1109/IAT.2006.23

[17] Audris Mockus. 2008. Missing Data in Software Engineering BT - Guide to Advanced Empirical Software Engineering. Springer London, London, 185–200. https://doi.org/10.1007/978-1-84800-044-5_7

[18] Audris Mockus. 2014. Engineering Big Data Solutions. In ICSE'14 FOSE. https://dl.acm.org/authorize?N14216

[19] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for Engineered Software Projects. Empirical Softw. Engg. 22, 6 (dec 2017), 3219–3253. https://doi.org/10.1007/s10664-017-9512-6

[20] Jerzy Neyman. 1992. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In Breakthroughs in Statistics: Methodology and Distribution. Springer, 123–150.

[21] Charles P. Quesenberry and Nicholas P. Jewell. 1986. Regression Analysis Based on Stratified Samples. Biometrika 73, 3 (1986), 605–614. http://www.jstor.org/stable/2336525

[22] Hitesh Sajnani, Vaibhav Saini, Joel Ossher, and Cristina V. Lopes. 2014. Is Popularity a Measure of Quality? An Analysis of Maven Components. In 2014 IEEE International Conference on Software Maintenance and Evolution. 231–240. https://doi.org/10.1109/ICSME.2014.45

[23] Software Heritage. 2022. Software Heritage. https://www.softwareheritage.org

[24] Zhongbin Sun, Jingqi Zhang, Heli Sun, and Xiaoyan Zhu. 2020. Collaborative filtering based recommendation of sampling methods for software defect prediction. Applied Soft Computing 90 (2020), 106163. https://doi.org/10.1016/j.asoc.2020.106163

[25] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. 2010. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In 2010 Asia Pacific Software Engineering Conference. 336–345. https://doi.org/10.1109/APSEC.2010.46

[26] Adam Tutko, Austin Henley, and Audris Mockus. 2020. More Effective Software Repository Mining. arXiv:2008.03439 [cs.SE]