Which Metrics Should Researchers Use to Collect Repositories: An Empirical Study

Kai Yamamoto[*], Masanari Kondo[*], Kinari Nishiura[*], Osamu Mizuno[*]
[*]Software Engineering Laboratory (SEL), Kyoto Institute of Technology, Japan
{k-yamamoto, m-kondo, k-nishiura}@se.is.kit.ac.jp, o-mizuno@kit.ac.jp

***Abstract*—GitHub is a huge publicly available development platform for hosting a version control system based on Git; software developers prefer to host their various software projects in GitHub. Therefore researchers who are interested in mining software repository frequently use GitHub to collect software projects as datasets. GitHub provides us with repository metrics such as popularity, contribution, and interest. We believe that such metrics are related to the quality of software; we use them to opt for studied repositories according to our research purpose. However, to the best of our knowledge, nobody has any evidence to support this assumption.**

Our main purpose is to provide researchers who study software quality, especially issue management, with *repository metrics* to select appropriate repositories for their studies. In this paper, we study the relationship between the characteristics of the issue pages of repositories that are selected by repository metrics in order to figure out the best repository metric to select proper repositories. The following findings are the highlights of our study: (1) The number of contributors that indicates the number of developers who contribute to a GitHub repository can be used to select the repositories having issue pages that are well-maintained. More specifically, such issue pages include more issues and in which developers use the labels more frequently rather than those that are selected by other metrics. (2) The number of dependencies opts for the repositories that have fewer issues and in which developers use the labels less often rather than those that are selected by other metrics.

***Index Terms*—issue; issue tracking system; GitHub; Libraries.io; repository;**

I. Introduction

The software development history enables software engineering researchers, especially mining software repository researchers, to study various topics such as defect prediction, code clone, and code smell. Hence, the system managing development history, Git, and the development platform hosting version control system based on Git, GitHub are frequently used by the researchers. More specifically, researchers collect repositories from GitHub as their studied datasets.

Since GitHub stores numerous repositories, we need to opt for repositories that are suitable for our research with *repository metrics*. For example, the number of *stars*, which is used to express our emotions such as interests, is frequently used to select repositories [5], [12], [23] since we believe that the number of stars is related to the software quality such as defect fixing and feature addition. However, to the best of our knowledge, nobody has evidence to support this assumption.

Vasilescu et al. [22] reported that as the number of either stars or *Forks* increases, the number of bug reports increases. In general, bug reports are sent to developers via *issues* on the *issue page* on GitHub. However, to the best of our knowledge, nobody studies the relationship between repository metrics, issues, and *labels* that indicates a domain for each issue.

Our main purpose is to provide researchers who study software quality, especially issue management, with *repository metrics* to select appropriate repositories for their research. In this paper, we study the relationship between the management processes in the issue page on GitHub and repository metrics. In addition, we report the difference of selected repositories across repository metrics. We conducted our experiments on projects that are publicly available in Maven [2] that is a package management system for Java and are also hosted on GitHub. In particular, we focus on the following research questions:

RQ1: **Do different repository metrics collect different GitHub repositories?**
Answer: Yes, they do.

RQ2: **Can repository metrics retrieve well-maintained repositories on GitHub?**
Answer: Yes, they can.

The following findings are the highlights of our study:

(1) The number of contributors that indicates the number of developers who contribute to a GitHub repository can be used to select the repositories having issue pages that are well-maintained. More specifically, such issue pages include more issues (at least about 10% larger) and in which developers use the labels more frequently rather than those that are selected by other metrics (at least about 8% larger).
(2) The number of dependencies opts for the repositories that have fewer issues compared to the number of contributors (less than over 80%) and in which developers use the labels less often rather than those that are selected by other metrics (less than over 10%).

Given the results, we recommend researchers to select repositories in terms of the number of contributors when researchers need repositories having enormous issues and in which developers use the labels frequently.

The organization of our paper is as follows. Section II describes key materials surrounding our research. Section III presents our research purpose and research questions. Section IV describes our experiments. Section V presents the results. Section VI discusses these results. Section VII discusses the threats to validity. Section VIII contextualizes our research. Section IX presents the conclusion.

II. Terminology

A. Libraries.io

Libraries.io [20] is a publicly available platform that collects and provides information about open source packages on 37 different package managers and three different development platforms (e. g., GitHub).

One of the metrics we retrieve in Libraries.io for each project (package) is *dependent projects count*. The dependent projects count indicates the number of projects that depend on the target project. Let us assume that we want to collect this metric from a `project A`. We would get the dependent projects count of two if two projects use the `project A`. We use this metric as a repository metric and we call it as *dependencies*. The number of dependencies corresponds to the dependencies that are written in `pom.xml` on Maven.

B. Maven

Maven is a package management system that is provided by Apache [2]. Maven mainly manages Java projects and addresses their dependencies with `pom.xml`. We write a pom.xml file for each project to configure the dependent package information.

C. GitHub

GitHub is a huge publicly available development platform for hosting a version control system based on Git; we can create private or public repositories on GitHub. GitHub provides metrics for each repository. In this paper, we use such metrics as *repository metrics*. We describe the metrics as follows:

- *Number of forks:* GitHub users can copy a repository into their own accounts by forking the repository [7]. They can modify the forked repository themselves without any affections to the original repository. In addition, they can suggest to merge their modifications to the original repository as a *pull request*.
- *Number of contributors:* A contributor is one who requests a change to a repository and changes the repository. In general, a contributor is added when the contributor forks a repository, makes a pull request, and such a pull request is merged to the master branch.
- *Number of stars:* GitHub users can add stars to repositories or topics. They can use their stars as a bookmark to such repositories or topics since we can easily track them on their *GitHub star pages* that list all the repositories that they add stars. In addition, adding a star to a repository shows our appreciation for developers who maintain the repository [11].
- *Number of watchers:* GitHub users can receive notifications (related to pull requests and issues) from watched repositories [9].

Prior studies [5], [6], [12], [15], [23] use the number of stars to select popular repositories. However, the number of forks, watchers, and contributors could also be used as metrics to select popular repositories. In addition, each of the metrics has different perspectives. Hence, we study not only the number of dependencies but also these four metrics (the number of forks, contributors, stars, and watchers) as the repository metrics.

To evaluate repository metrics, we use how much do developers maintain their repositories. We use the following two criteria as evaluation measures we called as *issue management criteria*:

- *Number of issues:* GitHub provides us with an issue tracking system [10]. GitHub users who have access right can create issues on a target repository. Issues are maintained on an *issue page* for each repository. An issue is used for various purposes such as bug reports, enhancements, and questions. An issue has a status that can be either open or closed; open indicates the issue is still work in progress; closed indicates the issue is already solved.
- *Number of labels:* Labels categorize issues and pull requests into groups in order to organize and prioritize them [8]. For example, we can use "bug" to be applied to issues that correspond to bug reports in the default setting. The repository owner can add new labels. Liao et al. [19] reported labeling improves the processing of issues. Hence, we use this number to evaluate the studied repository metrics.

We do not use these issue management criteria as repository metrics. This is because these issue management criteria are not explicitly shown for each repository. Hence if researchers want to use these criteria, they need to extract all issues from all the repositories. On the contrary, the repository metrics except for the dependency can be easily used in GitHub for researchers. For the dependency, all we need is to read `pom.xml`.

III. Research Question and Motivation

Our main purpose is to provide researchers who study issue management with *repository metrics* to select appropriate repositories for their studies. More specifically, we focus on a research topic in which researchers investigate the issue page in a repository and they want to collect well-maintained repositories on the issue pages. In this paper, we regard repositories that have high issue management criteria values as well-maintained repositories. To achieve our research purpose, we address two research questions.

A. RQ1: Do different repository metrics collect different GitHub repositories?

We would not need to pick up a repository metric if all repository metrics collect the same GitHub repositories. In this RQ, we study how much do different repository metrics collect different repositories.

B. RQ2: Can repository metrics retrieve well-maintained repositories on GitHub?

We would need to select a repository metric to collect studied repositories on GitHub if different repository metrics collect different repositories. In this RQ, we, therefore, compare GitHub repositories that were selected by different repository metrics in terms of the issue management criteria.

IV. Experimental Setup

In this section, we describe the studied dataset and experimental procedures.

A. Dataset

We use Java repositories since Java repositories are frequently used in software engineering research [14], [16]–[18]. In addition, we focus on the repositories that are managed by Maven and stored on GitHub since such repositories would be publicly available and useful. We use Libraries.io to find the repositories.

We use the Libraries.io open data version 1.4.0 that is the latest version when we start the experiment (Dec. 2019). This data, including 3,494,424 projects, was opened on Dec. 22, 2018. We retrieve 106,671 projects that are managed by Maven and stored on GitHub from the data. These projects belong to 31,429 repositories. Note that a repository can include multiple projects; and therefore, the number of projects and repositories are different.

Table I describes the studied data and their data source. We first extract top-500 repositories without overlap for each repository metric from the Libraries.io dataset, selecting such repositories by sorting them in descending order in terms of each repository metric. We secondly extract the number of forks, stars, watchers, closed issues, closed labeled issues and with or without issue page for each repository from GitHub via GitHub API v4[1]. We, therefore, extract the latest data from GitHub that do not exactly match the Libraries.io data. This is because (1) we need to use the data that are not provided by Libraries.io and (2) a bug of GitHub API restricts us extracting the data before the date of the Libraries.io data. All of them except for with or without issue page are integer values. With or without issue page indicates the presence of issue page for each repository. The issue page indicates this repository manages issues with GitHub. Third, we extract the number of contributors from the Libraries.io dataset because of the bug of GitHub API that affects the number of contributors. Finally, we extract the number of dependencies from the Libraries.io dataset since this is a unique data in the Libraries.io dataset.

B. RQ1 Approach

We use the top-500 repositories for each repository metric. If a repository has multiple projects, we use the max number of dependencies of a project as the number of dependencies of the repository. We compare the top-500 repositories between all possible pairwise combinations of the repository metrics. More specifically, we compare the duplicated repositories between such pairwise combinations.

[1]https://developer.github.com/v4/

C. RQ2 Approach

We use all the data in Table I. The studied repositories are the top-500 repositories for each repository metric. In this RQ, we execute four experiments:

- *Experiment 1. Issue Pages:* We count the number of repositories that have the issue page. In this experiment, we try to show the difference across repository metrics in terms of "with or without the issue page." This is important to check since some projects use other issue tracking systems such as JIRA. If we collect many repositories that use different issue tracking systems, researchers who investigate the issue page should avoid using the repository metric that is used to select the repositories.
- *Experiment 2. Usages of Issue Page:* For the repositories that have the issue page, we study the number of closed issues. If such repositories have at least one issue, we check whether such repositories use the label. In addition, we compute the proportion of labeled issues and the Pearson correlation coefficients between the number of closed issues and the number of labeled issues. We assume that the number of closed issues indicates how much do developers maintain this repository. If there are many closed issues, we consider this repository as well-maintained one since having many issues that are solved by developers. The number of labels is used to measure how many issues are categorized. In this experiment, we try to show the usage of the issue page functionals across different repositories that are selected by the repository metrics.
- *Experiment 3. A Bias by the Number of Issues:* We compute the Pearson correlation coefficients between the number of closed issues and lines of code (LOC) for all the repositories that are selected by the repository metrics and have the issue page. The number of closed issues of a repository might increase as the size of the repository is large. Hence, we use the correlation coefficients to check whether this bias exists. We use `cloc` [1] to count the sum of the number of LOC for each repository.
- *Experiment 4. A Correlation Analysis:* We compute the Pearson correlation coefficients between all possible pairwise combinations of the number of forks, contributors, stars, watchers, and the number of closed issues. If such combinations are strongly correlated with each other, we do not need to select a repository metric that is suitable for the experimental setting. Hence, we use the correlation coefficients to check whether they have differences.

V. Experimental Results

A. RQ1: Do different repository metrics collect different GitHub repositories?

Different repository metrics collect different repositories. Table II shows the proportion of matched repositories between all possible pairwise combinations of the repository metrics. The highest proportion is 75.2% between the number of forks and the number of watchers. This proportion is high; however,

TABLE I
THE DATA SOURCE

Data	Source	Description
Fork	GitHub	We extract the latest data with GitHub API v4.
Star	GitHub	The same as above.
Watcher	GitHub	The same as above.
With or Without Issue Page	GitHub	The same as above.
Closed Issue	GitHub	The same as above.
Closed Labeled Issue	GitHub	The same as above.
Top-500 repositories	Libraries.io	We extract top-500 repositories without overlap that are sorted by the values of a repository metric.
Contributor	Libraries.io	Because of the bug of GitHub API, we extract the data from Libraries.io.
Dependency	Libraries.io	We extract the data from Libraries.io since GitHub does not provide this data.

TABLE II
THE PROPORTION OF MATCHED REPOSITORIES ACROSS DIFFERENT REPOSITORY METRICS.

	# Contributors	# Stars	# Watchers	# Dependencies
# Forks	53.4%	73.4%	75.2%	16.6%
# Contributors	-	51.2%	47.0%	24.4%
# Stars	-	-	68.4%	12.8%
# Watchers	-	-	-	17.6%

TABLE III
THE PROPORTION OF REPOSITORIES THAT HAVE THE ISSUE PAGE

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
%IP	93.1%	88.1%	95.6%	94.2%	83.7%
#Error	5	4	1	3	4

24.8% of repositories are different even between the metrics that are the highest proportion pair. The lowest proportion is 12.8% between the number of stars and the number of dependencies. Although the number of forks, the number of stars, and the number of watchers result in around 70%, around 30% of repositories are different. The number of contributors and these three repository metrics result in around 50%; other combinations result in around 20%.

B. RQ2: Can repository metrics retrieve well-maintained repositories on GitHub?

Experiment 1. Issue Pages: Table III shows the proportion of repositories that have the issue page. %IP indicates the proportion; #Error indicates the number of repositories in which we met errors. The error reasons are (1) repository extraction error (e. g., the repository was already deleted) and (2) `cloc` error. When computing %IP, we exclude these repositories. In addition, we do not use such repositories in Experiment 2 and 3 as well. We describe this threat in Section VII-C.

Over 80% of the selected repositories by using any repository metrics have the issue page. The highest proportion is 95.6% of the number of stars. The number of dependencies has the lowest proportion; however, the proportion is still high (83.7%).

TABLE IV
THE 0TH (MIN), 25TH, 50TH, 75TH AND 100TH (MAX) PERCENTILES OF THE NUMBER OF CLOSED ISSUES

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
min	4.0	0.0	24.0	0.0	0.0
25th	368.0	579.0	306.0	216.8	37.0
50th	912.0	1,063.0	809.0	773.0	143.0
75th	2,049.0	2,193.0	1,762.0	1,899.5	553.5
max	20,655.0	20,654.0	20,678.0	20,665.0	20,137.0

TABLE V
THE PROPORTION OF REPOSITORIES THAT HAVE LABELED ISSUES IN THE REPOSITORIES THAT HAVE THE ISSUE PAGE

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
#hasIssueRepo	461	434	477	453	401
#hasLabel	434	427	455	417	321
#hasLabelPerAll	94.1%	98.4%	95.4%	92.1%	80.0%

Experiment 2. Usages of Issue Page: **The number of closed issues in the repositories that are selected by the number of contributors is the largest while selected by the number of dependencies is the smallest.** Table IV shows the percentiles of the number of closed issues. In the 25th, 50th, and 75th percentiles, the number of contributors shows the largest number of closed issues while the number of dependencies shows the smallest number of closed issues. The number of forks results in the second-largest median value.

TABLE VI
THE 0TH (MIN), 25TH, 50TH, 75TH AND 100TH (MAX) PERCENTILES OF THE NUMBER OF CLOSED LABELED ISSUES

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
min	0.0%	0.0%	0.0%	0.0%	0.0%
25th	11.4%	18.9%	11.9%	9.7%	1.6%
50th	28.1%	36.5%	27.3%	27.5%	22.2%
75th	50.4%	56.3%	47.7%	50.4%	54.0%
max	100.0%	100.0%	99.9%	100.0%	100.0%

TABLE VII
The Pearson correlation coefficients between the number of closed issues and the number of closed labeled issues

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
Coefficient	0.91	0.91	0.91	0.91	0.96

TABLE VIII
The Pearson correlation coefficients between the number of closed issues and LOC

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
Coefficient	0.161	0.145	0.424	0.160	0.160

The number of contributors uses labels most frequently while the number of dependencies uses labels most infrequently. Table V shows the proportion of repositories that have closed labeled issues. #hasIssueRepo indicates the number of repositories that have any issues; #hasLabel indicates the number of repositories that use any labels; #hasLabelPerAll indicates the proportion of #hasLabel in #hasIssueRepo. Indeed, #hasLabelPerAll of the number of contributors is 98.4% while #hasLabelPerAll of the number of dependencies is 80.0%. Table VI shows the percentiles of the number of closed labeled issues. In the 25th, 50th, and 75th percentiles, the number of contributors shows the largest number of closed labeled issues.

Table VII shows the Pearson correlation coefficients between the number of closed issues and the number of closed labeled issue. We observe that all repository metrics have over 0.91.

Experiment 3. A Bias by the Number of Issues: Table VIII shows the Pearson correlation coefficients between the number of closed issues and LOC. All the repository metrics except for the number of stars have low coefficients between 0.145 to 0.161. Even the number of stars results in 0.424.

Experiment 4. A Correlation Analysis: Table IX shows the number of studied repositories in this experiment. Since we do not use `cloc` in this experiment and our experiments are not affected by the error of `cloc`, the numbers of studied repositories increase.

Table X shows the Pearson correlation coefficients between all possible pairwise combinations of the repository metrics. For example, Fork-Contributors in the third column (the number of stars) is 0.551; this cell indicates the Pearson correlation coefficient between the number of forks and the number of contributors in the repositories that are selected by the number of stars. The underline indicates the correlation coefficients between the repository metric that is also used to select the repositories and the number of closed issues; the bold numbers indicate the largest numbers for each row.

TABLE IX
The number of repositories in Experiment 4

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
# repositories	463	439	477	468	416

We observe the following findings:

- Fork-Star, Fork-Watcher, and Star-Watcher result in over 0.80 in any repositories sets that are selected by the repository metrics.
- Any repository metrics pairs in the repositories that are selected by the number of dependencies result in larger correlation coefficients compared to the correlation coefficients that are computed in the other repositories sets.
- The number of closed issues is correlated (over 0.7) with the number of forks, contributors, and watchers in the repositories that are selected by the number of dependencies; the number of closed issues is correlated (over 0.7) with the number of contributors in the repositories that are selected by the number of watchers. However, the other repository metrics do not have high correlation coefficients (less than 0.7).
- All the underlined values are over 0.60.

VI. Discussion

A. RQ1: Do different repository metrics collect different GitHub repositories?

The numbers of forks, stars, and watchers result in 70% matched rates; however, around 30% of repositories are different. The matched rates of the other combinations are lower than 70%. Hence, each repository metric selects different repository sets. Table XI shows the number of unique repositories that are selected for each metric. We look at the selected repositories for each repository metric.

The number of dependencies: A part of the repositories the number of dependencies only detects is the following repositories:

- `mysql/mysql-connector-j`
- `webjars/jquery`
- `jboss/jboss-parent-pom`

Since `mysql/mysql-connector-j` is an old project and has already reached maturity, this project has only released minor changes in the last two years. In such a project, we do not need to carefully watch the releases to check changes such as adding new features. In addition, since developers do not need to add new features, the number of forks and contributors are also low. This might be a reason why this repository is only detected by the number of dependencies.

`webjars/jquery` and `jboss/jboss-parent-pom` have few source codes. These repositories have configurations in order to use repositories, software, and libraries from other ones. Since these repositories have few source codes, there rarely exist bugs and enhancements. This might be a reason why this repository is only detected by the number of dependencies.

Borges et al. [4] reported related results. They found that over half of users add a star to a repository since (1) users want

TABLE X
The Pearson correlation coefficients across repository metrics

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
Fork-Contributors	0.527	0.550	0.551	0.568	**0.687**
Fork-Star	0.812	0.825	0.823	0.821	**0.882**
Fork-Watcher	0.908	0.917	0.914	0.872	**0.923**
Fork-ClosedIssue	0.653	0.663	0.685	0.673	**0.763**
Contributors-Star	0.544	0.539	0.557	0.591	**0.745**
Contributors-Watcher	0.497	0.526	0.520	0.483	**0.732**
Contributors-ClosedIssue	0.677	0.622	0.699	0.702	**0.753**
Star-Watcher	0.896	0.914	0.900	0.840	**0.956**
Star-ClosedIssue	0.608	0.616	0.641	0.632	**0.666**
Watcher-ClosedIssue	0.644	0.666	0.677	0.617	**0.726**

TABLE XI
The number of unique repositories that are selected for each metric.

# Forks	# Contributors	# Stars	# Watchers	# Dependencies
44	137	73	80	349

to use the repository as a reference to their own repositories and (2) users are interested in the repository; under 40% of users add a star to a repository since users use the repository. Hence, the number of stars of a repository and the number of users who use the repository (dependencies) are not related.

The number of watchers: A part of the repositories the number of watchers only detects is the following repositories:

- `uber/react-map-gl`
- `wix/wix-restaurants-availability`
- `Netflix/Priam`

`uber/react-map-gl` has frequently released major versions. For `wix/wix-restaurants-availability`, the number of stars is 2 while the number of watchers is 352 on Feb. 2, 2020. In addition, the usage of this repository in the README file has not been written yet. Hence, this repository should be at the early stage of the development. This result implies that users watch a repository if such a repository would be updated. In addition, the repositories the number of watchers only detects include 29 Netflix repositories and 9 Uber repositories. This result implies that users are interested in the repositories that are developed by popular companies even at the early stage of the development.

The number of forks: A part of the repositories the number of forks only detects is the following repositories:

- `BlackrockDigital/startbootstrap-simple-sidebar`
- `mathiasbynens/jquery-placeholder`
- `bingoogolapple/BGARefreshLayout-Android`

`BlackrockDigital/startbootstrap-simple-sidebar` has only one contributor. The other repositories also have relatively few contributors. Since the number of forks of these repositories should be large, users who fork these repositories do not contribute the repositories. Hence, users want to fork the repositories in order to not modify the repositories but use them for other projects.

The number of stars: A part of the repositories the number of stars only detects is the following repositories:

- `kristoferjoseph/flexboxgrid`
- `svgdotjs/svg.js`
- `lipis/flag-icon-css`

`kristoferjoseph/flexboxgrid` is a tool to draw using CSS and supports users to easily prepare their own web pages. The other repositories are also related to animation and CSS. Hence, Web and UI might be a motivation to add stars to a repository. In addition, these repositories have moderate numbers of forks. Therefore, if we use more than 500 repositories (e. g., 1,000), the matched rate between the number of forks and stars would increase.

The number of contributors: A part of the repositories the number of contributors only detects is the following repositories:

- `gatling/gatling`
- `qunitjs/qunit`
- `kulshekhar/ts-jest`
- `palantir/tslint`
- `languagetool-org/languagetool`
- `scalameta/scalafmt`

`gatling/gatling`, `qunitjs/qunit`, and `kulshekhar/ts-jest` are tools related to software test. This is because the test tools are updated by external effects. Let us assume that if a target software A is updated and it is difficult to test the new version using the current test tool B. Developers of A would make a new pull request to address this problem. Hence, this result implies that external developers make pull requests more frequently on the repositories that are related to test tools rather than other repositories and the number of contributors also increases. `palantir/tslint`, `languagetool-org/languagetool`, and `scalameta/scalafmt` are related to language and formats. These tools are frequently written by using the rule-based technique; adding a new rule is easier for developers. Hence the number of contributors increases.

From the results and discussions, we answer the RQ1.

The different repository metrics collect different repositories. Hence, if we use a repository metric to collect repositories, we need to consider a threat that the collected repositories are biased.

B. RQ2: Can repository metrics retrieve well-maintained repositories on GitHub?

Experiment 1. Issue Pages: The number of stars is the most suitable for selecting the repositories that have the issue page while the number of dependencies is not (Table III). The repositories that do not use the issue page use a bug tracking system instead. The repositories that have a long history tend to have more dependencies and use a bug tracking system because GitHub is a new platform compared to bug tracking systems. However, the lowest proportion is still over 80%. Hence, any repository metrics could collect repositories that have the issue page.

Experiment 2. Usages of Issue Page: The number of contributors has the largest median value in terms of the number of closed issues (Table IV). However, the proportion of the number of repositories that have the issue page is lower than the number of forks, stars, and watchers (Table III); they have different conditions.

The number of closed issues in the repositories that are selected by the number of contributors is still the largest. To relieve the different conditions across the repository metrics, we extract only top-437 repositories for each repository set that was selected by either the number of forks, stars, or watchers; 437 is the number of repositories that were selected by the number of contributors and have the issue page. We use 437 since this is the lowest value across all the repository metrics except for the number of dependencies. In this comparison, we exclude the number of dependencies since the number of closed issues on the repository set that was selected by the number of dependencies is certainly small (Table IV). Table XII shows the percentiles of the number of closed issues on the top-437 repositories. We observe that the number of contributors still has the largest numbers of closed issues on 25th, 50th, and 75th percentiles.

The number of closed labeled issues in the repositories that are selected by the number of contributors is still the largest. We have the same concern as the result of the labeled issues. To relieve the different conditions across the repository metrics, we extract only top-401 repositories for each repository set that was selected by the number of forks, contributors, stars, and watchers; 401 is the number of repositories that were selected by the number of dependencies and have at least one labeled issue. We use 401 since this is the lowest value across all the repository metrics. We compute the proportion of labeled issues on the top-401 repositories. Table XIII shows the percentiles of the number of closed labeled issues on the top-401 repositories. We observe that the number of contributors has the largest number of closed labeled issues on 25th, 50th, and 75th percentiles.

TABLE XII
THE 0TH (MIN), 25TH, 50TH, 75TH AND 100TH (MAX) PERCENTILES OF THE NUMBER OF CLOSED ISSUES ON THE TOP-437 REPOSITORIES

	# Forks	# Contributors	# Stars	# Watchers
min	4	0	24	0
25th	376.0	579.0	337.0	230.0
50th	946.0	1,063.0	864.0	811.0
75th	2,075.0	2,193.0	1,883.0	1,957.0
max	20,655	20,654	20,678	20,665

TABLE XIII
THE 0TH (MIN), 25TH, 50TH, 75TH AND 100TH (MAX) PERCENTILES OF THE NUMBER OF CLOSED LABELED ISSUES ON THE TOP-401 REPOSITORIES

	# Forks	# Contributors	# Stars	# Watchers	# Dependencies
min	0.0%	0.0%	0.0%	0.0%	0.0%
25th	11.4%	20.1%	12.8%	12.3%	1.6%
50th	28.1%	37.7%	28.5%	29.9%	22.2%
75th	50.4%	56.6%	47.8%	50.9%	54.0%
max	100.0%	100.0%	99.9%	100.0%	100.0%

On the contrary, the repositories selected by the number of dependencies has the smallest number of closed issues.

The number of closed issues and the number of labeled closed issues are strongly correlated (Table VII).

These results imply that:

- The number of dependencies has the smallest number of closed issues; and therefore, developers do not need to classify them with labels since the number of issues might also be small.
- The number of contributors has the largest number of closed issues; and therefore, developers need to classify them with labels since the number of issues might also be large.

Hence, the number of contributors selects the repositories that have more closed labeled issues rather than the other repository metrics.

Experiment 3. A Bias by the Number of Issues: We observed that the number of closed issues and LOC are not correlated (Table VIII). This result implies that the number of closed issues is regardless of the size of repositories; instead, the number of closed issues is related to the maintenance effort.

Experiment 4. A Correlation Analysis: The numbers of forks, stars, and watchers select similar repositories in Experiment 1. We observed the reason in this experiment that such repository metrics are strongly correlated (over 0.8). Sheoran et al. [21] have already reported that the number of watchers and forks are strongly correlated. Our experiments report additional relationships. We also observed that almost all combinations are moderately correlated.

From the results and discussions, we answer the RQ2.

The number of contributors is the best repository metric to collect the repositories that have more closed labeled issues (at least about 8% larger). The numbers of forks, stars, and watchers select similar repositories. The number of dependencies selects the repositories that have fewer issues compared to the number of contributors (less than over 80%).

C. Recommendation

Our experimental result shows that different repository metrics opt for different repositories. The number of contributors selects the repositories that have more closed issues and labeled issues compared to the other repository metrics. In addition, the number of contributors and the number of closed issues are moderately correlated. Hence, we recommend researchers who need issues to use the number of contributors in order to select studied repositories.

VII. Threats to Validity

A. External validity

We picked up a part of repositories (500 repositories) that were selected by the repository metrics, executed the experiments on the repositories, and discussed the results. It is difficult and time-consuming to collect all repositories on GitHub because of the rate limit of GitHub API. Future studies are necessary to investigate whether our discussions apply to other repositories that were not included in the 500 repositories.

B. Internal validity

We provide the experimental scripts that we used to collect repository metrics from GitHub[2]. The experimental scripts and the Libraries.io dataset (version 1.4.0) allow researchers and practitioners to replicate our experiments and confirm our results.

C. Construct Validity

We collected the number of contributors from Libraries.io. On the contrary, we collected the number of closed issues, closed labeled issues, forks, stars, and watchers from GitHub with GitHub API. This is because (1) Libraries.io does not provide the number of closed issues and closed labeled issues and (2) a bug of GitHub API restricts us extracting the data before the date of the Libraries.io data. We manually looked at the number of contributors on GitHub and Libraries.io and compared them. This analysis results in that they are not significantly different. Hence, using both Libraries.io and GitHub is acceptable.

We met errors when conducting the RQ2 experiments (Table III). We extract 500 repositories; the number of repositories in which we met errors is between 1 to 5. The error rate is less than or equal to 1%. The impact of these errors on our findings and results is little.

We only use the issue page on GitHub as the studied issue management system. On the contrary, the Apache Software Foundation, for example, uses JIRA, which is an issue management system, instead of the issue page. Future studies are necessary to consider the other issue management systems.

The combination of repository metrics might select different sets of repositories. Future studies are necessary to investigate the combination of repository metrics.

VIII. Related Work

Sheoran et al. [21] studied the relationship between developers who are watchers and contributors. They found that a few developers who are watchers become contributors. In addition, contributors who were watchers contribute to a repository for a longer time compared to the other contributors. Borges et al. [4] reported that the number of stars significantly increases when conducting marketing activities. In addition, about 50% of users regard stars as a bookmark. Borges et al. [3] reported that the number of stars and forks are strongly related and the number of stars and contributors are weakly correlated. These results are consistent with our results.

Prior research work that studied the quality of repositories or files used the number of stars to select popular repositories [5], [6], [12], [15], [23]. However, we can use various repository metrics to select popular repositories such as the number of contributors. Our experimental results would support to select a suitable repository metric to select studied repositories.

Gyimesi et al. [13] used labels that are related to bugs in order to clarify the characteristics of bugs. Our experiment also studied the labels of issues. Hence, our results could be used when conducting such researches.

IX. Conclusion

We studied the repository metrics in order to clarify the best-performing metric to select repositories that have many more closed issues and closed labeled issues. In addition, we compute the Pearson correlation coefficients between all possible pairwise combinations of the repository metrics, the number of closed issues, and the number of closed labeled issues.

Our experimental results show that the number of contributors collects the repositories that have many more closed issues and closed labeled issues compared to the other studied repository metrics. Hence, researchers who study software quality based on issues should use the number of contributors to select studied repositories.

Acknowledgment

This work has been supported by JSPS KAKENHI Japan (Grant Numbers: JP19J23477).

References

[1] AlDanial. Aldanial/cloc. [Online]. Available: https://github.com/AlDanial/cloc

[2] Apache. Apache maven project. [Online]. Available: https://maven.apache.org/

[2]https://github.com/LakeRainSound/get_code_metrics

[3] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 334–344.

[4] H. Borges and M. T. Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112 – 129, 2018.

[5] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An empirical analysis of the docker container ecosystem on github," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 323–333.

[6] E. Cohen and M. P. Consens, "Large-scale analysis of the co-commit patterns of the active developers in github's top repositories," in *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*. Association for Computing Machinery, 2018, p. 426–436.

[7] GitHub. About forks. [Online]. Available: https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-forks

[8] ——. About labels. [Online]. Available: https://help.github.com/en/github/managing-your-work-on-github/about-labels

[9] ——. About notifications. [Online]. Available: https://help.github.com/en/github/receiving-notifications-about-activity-on-github/about-notifications

[10] ——. Managing your work with issues. [Online]. Available: https://help.github.com/en/github/managing-your-work-on-github/managing-your-work-with-issues

[11] ——. Saving repositories with stars. [Online]. Available: https://help.github.com/en/github/getting-started-with-github/saving-repositories-with-stars\#about-stars

[12] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 352–355. [Online]. Available: https://doi.org/10.1145/2597073.2597118

[13] P. Gyimesi, G. Gyimesi, Z. Tóth, and R. Ferenc, "Characterization of source code defects by data mining conducted on github," in *Computational Science and Its Applications (ICCSA)*. Springer International Publishing, 2015, pp. 47–62.

[14] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 200–210.

[15] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, *GitHub Projects. Quality Analysis of Open-Source Software*. Cham: Springer International Publishing, 2014, pp. 80–94. [Online]. Available: https://doi.org/10.1007/978-3-319-13734-6_6

[16] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2012.

[17] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[18] M. Kondo, D. M. German, O. Mizuno, and E.-H. Choi, "The impact of context metrics on just-in-time defect prediction," *Empirical Software Engineering*, vol. 25, no. 1, pp. 890–939, 2020.

[19] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[20] Libraries.io. Libraries.io. [Online]. Available: https://libraries.io/

[21] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian, and J. Ell, "Understanding "watchers" on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. Association for Computing Machinery, 2014, p. 336–339.

[22] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 805–816. [Online]. Available: https://doi.org/10.1145/2786805.2786850

[23] Y. Wu, J. Kropczynski, R. Prates, and J. Carroll, "Understanding how github supports curation repositories," *Future Internet*, vol. 10, p. 29, 03 2018.