Empirical Distribution of Code Reviews in DHIS2 Repositories and the Impact of Code Reviews on the Popularity of DHIS2 Repositories on GitHub

Leonard Peter Binamungu, and Jimmy T. Mbelwa
Department of Computer Science and Engineering,
College of Information and Communication Technologies,
University of Dar es Salaam, Dar es Salaam, Tanzania
{lepebina@udsm.ac.tz, jimmymbelwa@gmail.com}

***Abstract*—The District Health Information Software Version 2 (DHIS2) facilitates the collection and analysis of routine health data, to inform planning and decision making in the health sector. Majority of previous studies on DHIS2 have focused on social and political factors affecting the adoption of DHIS2 in different countries. However, the software engineering aspects of the DHIS2 have not been widely studied. Code review enables developers to review each other's code to detect quality issues and share knowledge. Studies have investigated factors affecting the popularity of repositories in social coding websites like GitHub. However, as far as we know, no previous studies have investigated the impact that code reviews could have on the popularity of repositories on social coding websites. To make a start in filling this gap, we analysed 117 DHIS2 GitHub repositories to gather empirical evidence about the distribution of code reviews in these repositories, as well as the relationship between code review and the popularity of DHIS2 repositories on GitHub. We found the presence of minimal code review in DHIS2 repositories. Moreover, code review was generally found to affect the popularity of DHIS2 repositories on GitHub.**

***Index Terms*—Code review, Source Code Popularity, Open Source Software, District Health Information Software, DHIS2**

I. Introduction

The DHIS2 [1] is an open-source software currently used by more than 60 countries across the world (mostly in Africa and the developing world) [2, 3] to collect, analyse and visualise routine health data across the country, to inform planning, policy and decision making. In Tanzania, for example, the DHIS2 has been endorsed by the health ministry as one of the official sources of government health data. DHIS2 has evolved over time to cover data needs of various stakeholders, even beyond the health context [1, 4].

The DHIS2 is developed and maintained by the Health Information System Programme at the University of Oslo [3, 5], but it also attracts global contributions, especially from developers responsible for maintaining country-specific DHIS2 instances. Given its wide use in low-and-middle-income countries, it is important to study the DHIS2 development processes, to uncover lessons that could contribute to its continuous improvement and sustainability. Most of previous studies on DHIS2 have focused on social and political factors affecting its adoption in different countries (e.g., [6]–[8]). But the software engineering aspects of the DHIS2 have not been widely studied.

Code review is the process in which one developer's code is analysed by peers to detect and correct quality issues and arrive at common code understanding [9]–[15]. Different from the traditional rigorous formal code inspections, modern code review adopts a lightweight review process in which developers can interact and exchange comments about written code [10, 16]. Moreover, factors that affect the popularity of code repositories on social coding websites have been studied [17]–[20].

However, to the best of our knowledge, previous studies have not attempted to understand the impact that code review can have on the popularity of repositories in social coding websites like GitHub. To make a start in filling this gap, we analysed 117 public DHIS2 repositories on GitHub, to understand the prevalence of code reviews amongst DHIS2 repositories on GitHub, and the impact of code reviews on the popularity of DHIS2 repositories on GitHub. We found the presence of minimal code review activity in DHIS2 repositories, and minimal evidence of code improvement based on code review. Moreover, code review was generally found to affect the popularity of DHIS2 repositories on GitHub. We, thus, contribute to the literature on how code reviews are distributed in Open Source Software (OSS) systems, and to the understanding of the relationship between code reviews and the popularity of code repositories.

II. Related Work

Importance of code reviews in source code repositories: Code review is an important aspect in software development–it is employed by developers in both open-source and proprietary settings, and it involves the review of developers' code by peers. McIntosh *et al.* [9] provide evidence that code review reduces the number of bugs detected after release as long as the code review coverage is adequate and there is good participation of team members–code authors and reviewers–in the code review process. Hundhausen *et al.* [21] conducted a study to understand the impact of code reviews on student learning and soft skills. Their results revealed that code reviews can benefit both source code authors and reviewers through knowledge exchange. Other authors have shown that code review improves collaboration among the team because it develops collective source code authorship [14, 22].

Empirical distribution of code reviews in source code repositories: Studies investigating different aspects of code reviews have used several datasets that can shed light on the prevalence of code reviews in source code repositories. Baysal *et al.* [23] performed an empirical study of the WebKit code review process by focusing on non-technical factors such as priority, patch size and component size, as well as organisational and personal dimensions (experience of patch writer, reviewer load and activity). Data extracted from WebCore were 10,012 patches that contained file changes and that had been reviewed. Their findings provide empirical evidence that personal and organizational factors have a significant influence in the code review process regarding the review timeliness as well as the chance of patch acceptance.

Thongtanunam *et al.* [16] empirically investigated the challenge of allocating appropriate reviewers to code, and the impact of reviewer allocation problem on code review time. Four open-source projects namely Android Open Source Project, Qt, OpenStack and LibreOffice projects were used in their study. They manually examined 7,597 comments from 1,461 reviewed samples; and then 42,045 reviews were examined using file location-based code-reviewer recommendation tool of four open software systems. The results show that the time to review and approve code changes takes longer but can be reduced with tools support helping developers to assign code to appropriate code reviewers and hence speed up the code review process.
In another study, Thongtanunam *et al.* [10] investigated the influence of modern code review in clean and defective source code files. Data was collected from Qt open source project, and 11,736 reviews of changes to 24, 486 files were empirically studied. The findings show that thorough code review might reduce the probability of future defects.

Popularity of open source repositories and software applications: Zhu *et al.* [17] studied the frequency of folders used by 140,000 GitHub projects, and found that the use of standard folders (e.g. test, doc, examples) increase the probability of the code to be forked and therefore may have an impact on project source code popularity. Borges *et al.* [18] investigated the popularity of the GitHub repositories and discovered that application domains and programming language influence the number of stars of GitHub projects. Moreover, the study revealed that repositories owned by organisations are more popular than individuals' repositories, and that forks influence the number of stars on repositories.

Lu et *et al.* [19] mined a taxonomy of 108 features of 9824 apps from Wandoujia–a Chinese Android App store, and found that user rating and downloads were related to apps popularity. Businge *et al.* [20] examined how social and technical factors mined from Github and Google Play store relate to Apps popularity, and discovered that both social and technical factors significantly describe App popularity. They, however, found that the combination of technical and social factors have small effect on the popularity of Apps on Google Play.

Research gap: To the best of our knowledge, no previous studies have investigated the relationship between code reviews and the popularity of repositories on social coding websites like GitHub.

III. Research Design

A. Dependent and Independent Variables

Dependent Variables: Several metrics for measuring the popularity of source code repositories have been proposed in the literature [18, 20, 24]. To make a start in exploring the relationship between code reviews and the popularity of source code repositories, in the present study, we chose to use the following three metrics to measure the popularity of DHIS2 repositories on GitHub, all of which have been used by previous studies [20, 24] to measure the popularity of software repositories on GitHub:

- *Forks:* This measures the total number of forks accumulated by a repository on GitHub. In GitHub and other social coding websites, if a developer wants to to develop on top of a particular repository, they often fork it and start a new development line. If they discover and fix bugs in the repository, or add new functionality to that repository, they may wish to contribute back to the main repository through Pull Request or issue reporting, thereby contributing to strengthening the main repository [20]. A highly forked repository indicates strong interest by developers beyond its main development line, and so stands good chances of sustainability, especially if forking developers contribute back [20, 25]. The opposite is also true–a less forked repository is less popular among developers in the social coding community.
- *Active Forks:* This represents forks with at least one commit since the forking date [20]. A higher number of commits on an Active Fork means that developers are actively working on the fork, and could potentially contribute back to (and hence improve) the main repository. Lower numbers of commits could suggest that the fork has been abandoned, for various reasons, some of which could be quality related.
- *Stars:* If developers on a social coding website like a repository, they will sometimes star it to indicate interest and/or how useful the repository was to them [24]. While more stars could indicate strong interest, fewer stars could indicate little interest in the repository by the social coding developer community.

Independent Variables: We devised three metrics to quantify the extent of code review in a particular repository:

- *Total Comments:* This represents the total number of code review comments in a particular repository. It is similar to *Discussion Length* [10, 26, 27]. Whereas more comments on different commits would indicate that developers discuss the contributions by various members to identify areas of improvement, little to no comment would suggest that developers do not discuss code from various members, and that could lead to more bugs and issues, negatively impacting the quality of code in a repository in the long run.

- *Instances of Code Review:* Number of code fragments involved in code review. In this context, a fragment could be a line, statement, block, method, class, Pull Request, etc. Higher values of review instances indicate that more code fragments in a repository have been reviewed, while lower values of review instances suggest that large chunks of code in a repository have not been reviewed, and that could negatively impact the quality of a repository [10, 28].
- *Instances of Effective Code Review:* Number of code review instances in which the review feedback was effected by code authors. Because not all recommendations made during code reviews get implemented by developers of the reviewed code, it is good to discriminate between reviews that were effected and reviews that were not effected. Higher numbers of effective code review instances could lead to code improvement.
- *SLOC:* Repository size in terms of lines of code. We use SLOC as a *control variable*, because code repositories of different sizes are likely to have different numbers of code review comments, code review instances, and effective code review instances.
- *Commits:* Number of commits on a master branch[1]. This is also used as a *control variable* because code review is often done on specific commits, and so repositories with higher numbers of commits are likely to have more evidence of code review activity than repositories with lower numbers of commits.

B. Data Collection

Data for this study were extracted from public DHIS2 GitHub repositories, under the *dhis2* account on Github[2]. Our analysis covers the state of the repositories up to and including 20th March, 2020; 142 public repositories were part of the dhis2 GitHub account by that date. Similar to the work of Businge *et al.* [20], the GitHub REST API[3] was used to extract information for code review and popularity metrics discussed in section III-A, and *cloc*[4] was used to determine SLOC for each repository.

Starting with an initial list of 142 repositories, 25 repositories were excluded from further analyses based on the following five criteria (the number of repositories removed based on a particular criteria are shown in brackets): **One**, repositories with only database files, as well as repositories that were only meant for archiving of source code and other information about various DHIS2 projects (2). Such repositories are likely to have no the kind of developer activity we were interested in, and manual analysis confirmed this intuition. **Two**, repositories created only for training purposes (5). Their removal aimed to minimise the possibility of biasing results with trivial demonstration projects. Because training/demonstration repositories are often created to serve short term knowledge sharing purposes, one would not expect them to have the kind of code review rigour this study was interested in. **Three**, repositories with less than 5 commits on a master branch (11). Manual analysis of the branches of such repositories revealed that they also resembled trivial projects in criteria two. **Four**, repositories which were just about documenting acceptable GitHub practices for DHIS2 developers (1). **Five**, repositories in which *cloc* was unable to compute values for SLOC (6). This left us with 117 repositories which were subjected to further analysis. The results reported hereafter are based on this collection of 117 repositories.

For each of the 117 repositories, we conducted manual analysis of comments in code in order to get values for the three code review metrics introduced in section III-A. For a particular repository, the value of *Total Comments* was obtained from the number of all the comments returned by the GitHub REST API. To get the value of *Instances of Code Review*, we manually analysed the comments to identify the number of different code fragments covered by the review.

In modern code review processes, code authors can address reviews by responding to comments or producing new versions of code that take proposed changes into account [9]. We restricted our measure of code review effectiveness to the instances of code where authors responded to comments made on their code, by clarifying their position to the persuasion of reviewers or reporting that issues raised in the review had been fixed. Thus, to get the value of *Instances of Effective Code Review* for a particular repository, we counted the number of reviewed code fragments in which authors of the reviewed fragments either accepted the reviews and agreed to make changes on their code, or had persuaded reviewers to agree with initial design decisions, or said explicitly in comments that they had made changes in code based on review feedback. Code review not only enables developers to receive feedback about their code, but also it enables knowledge sharing among team members [14] where various decisions made by individual developers can be debated, and reaching consensus in such debates could increase code ownership across team members [21, 22], adding to quality and long term sustainability of the software. We, therefore, also count a review in which a reviewer is persuaded by an author of the code to be an effective instance of code review.

C. Model Building

We constructed multiple linear regression models to understand if there is a relationship between our independent variables (code reviews aspects) and the dependent variables

[1]While code review can be done on any branch, we decided to focus on master branch because, for most development teams, it is unlikely to merge a feature branch to master without code review.

[2]https://github.com/dhis2/

[3]https://developer.github.com/v3/

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

(popularity of DHIS2 repositories on GitHub). A multiple linear regression model takes the form $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$, where y is a dependent variable and every x_i represents a particular independent variable. Because both the present work and the work of McIntosh *et al.* [9] aimed at understanding the relationship between some code review facets and some dependent variables, we adapted the model building approach used by McIntosh *et al.* [9]. Skewness tests revealed that the values for all our variables were *highly skewed*[5], and so we performed $log(x+1)$ transformation on all our variables, to minimise the effect of outliers, stabilise variance and enhance model fit.

To minimise multicollinearity, we used Spearman rank correlation coefficient (ρ) to test for highly correlated independent variables. Spearman rank correlation was used because it is insensitive to not-normally-distributed data. Similar to the work of McIntosh *et al.* [9], $\mid \rho \mid > 0.7$ was used to detect highly correlated variable pairs and, if a variable pair was found to be highly correlated, only one of a pair was included in a model. Based on this process, we remained with the following three independent variables which are the ones we used to construct the model: Commits, Total Comments, and Effective Instances.

Moreover, the models built out of the remaining three variables were also checked for multicollinearity using the Variance Inflation Factor (VIF). In particular, we computed a VIF value for each independent variable used in model construction. While a VIF value of 1 suggests that a variable is not correlated with others, a VIF $>$ 1 indicates the variance inflation ratio captured by the model as a result of collinearity. Similar to McIntosh *et al.* [9], we use five as a VIF threshold, and all the three variables in the models were found to have VIF values less than five, and so there was no further removal of any variable.

IV. Results and Discussion

A. Descriptive statistics for the studied DHIS2 repositories

Table I shows the descriptive statistics of the DHIS2 repositories we analysed. For each variable, the mean is greater than the median, indicating that all the variables are positively skewed. Moreover, the statistics in Table I indicate that a lot of the analysed repositories were of sizeable lines of code and number of commits. As well, apart from displaying evidence of generally minimal code review, a fairly large number of the analysed repositories had been forked and "stared" several times.

TABLE I
Descriptive statistics of the DHIS2 repositories involved in the analysis

S/n	Metric	Min	First Quartile	Median	Mean	Thrid Quartile	Max
1	SLOC	9	688	2308	22193	10683	648884
2	Commits	5	34	134	429.8	453	9367
3	Total Comments	0	0	0	0.8205	0	29
4	Review Instances	0	0	0	0.5897	0	22
5	Effective Instances	0	0	0	0.2222	0	7
6	Forks	0	0	2	6.667	5	173
7	Active Forks	0	0	1	1.949	2	18
8	Stars	0	0	1	4.248	3	156

Different DHIS2 repositories offer different functionalities. The majority (83) of the repositories we analysed focused on supporting the work of DHIS2 developers through such things as providing tools for developing Android mobile apps for use on the DHIS2 platform[6], User Interface design tools for DHIS2 apps[7], and other functions such as facilitating data pre-processing[8] before generating a variety of reports required by end users. Another category of the analysed repositories (25) consists of apps for enabling end users to perform some functions on the DHIS2 platform. DHIS2 is mainly organised as a collection of apps that enable users to perform various tasks over the platform–for example, data entry[9], quality checks[10], reports generation, data visualisations, data analysis with Pivot Tables[11], GIS Maps, and many more. Finally, 9 of the repositories we analysed combined both developer support and end-user application functions.

B. Empirical distribution of code reviews in DHIS2 repositories

Table II shows the distribution of code reviews across the analysed DHIS2 repositories. Based on these results, generally, only a fifth of the repositories we analysed had evidence of code reviews. Furthermore, out of all code instances that were found to have been reviewed, in only 37.68% of those instances did the review lead to code improvement.
Figure 1 shows the distribution of reviews in the DHIS2 reviewed code. It can be seen that majority of DHIS2 repositories with evidence of code reviews have less than five code review comments. Also, in most repositories with evidence of code review, the number of instances of code involved in the review is less than five. Moreover, majority of effective code review instances are less than five, implying that most of the code review did not lead to code improvement.

These results, in general, suggest that only a very small proportion of DHIS2 code in public repositories is reviewed. However, it could well be that, generally, DHIS2 code reviewers do not comment on code if they have nothing important to say about it. That could hinder the measure of code review activity based on the availability of comments in code. Nevertheless, further investigation into how DHIS2 developers approach code review in practice could shed more light on this. Also, the possible reasons for lack of evidence of code improvement based code reviews can be related to the work of Bacchelli and Bird [14], in which it was empirically observed

[5]https://rpubs.com/marvinlemos/log-transformation
[6]https://github.com/dhis2/dhis2-android-sdk.git
[7]https://github.com/dhis2/ui-forms.git
[8]https://github.com/dhis2/usage-analytics-app.git
[9]https://github.com/dhis2/dhis2-android-dataentry.git
[10]https://github.com/dhis2/who-data-quality-app.git
[11]https://github.com/dhis2/pivot-tables-app.git

TABLE II
DISTRIBUTION OF CODE REVIEWS IN DHIS2 REPOSITORIES

Item	Description	Amount
A	Number of repositories analysed	117
B	Number of repositories with evidence of code reviews	24
C	Number of repositories without evidence code reviews	93
D	Number of code review comments across the analysed repositories	96
E	Number of instances of code involved in the review, across the analysed repositories	69
F	Number of effective code reviews, out of code instances involved in review	26
G	Percentage of repositories with evidence of code reviews (B*100/A)	20.51%
H	Percentage of effective code review instances (F*100/E)	37.68%

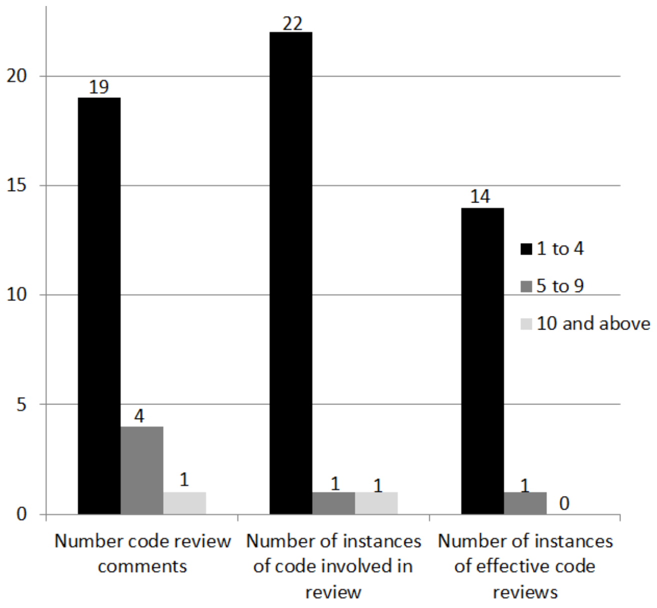

Fig. 1. Distribution of reviews in DHIS2 reviewed code

that understanding of the reasons for code change and the code itself are key aspects in the code review process for both source code authors and reviewers. If reviewers do not understand why certain changes were made in code, it may be hard for them to review it properly; similarly, if developers lack clarity about the changes requested by reviewers, it may be difficult for them to improve their code based on code reviews. However, measuring the effectiveness of code reviews by analysing later commits by developers of reviewed code, and studying the actual DHIS2 code review workflow, could give more insights into whether code review lead to code improvement in DHIS2 repositories.

C. Impact of code reviews on the popularity of repositories on GitHub

Table III shows the relationship between code review and popularity of DHIS2 repositories on GitHub. The leftmost column represents the dependent variables–Forks, Active Forks, and Stars–which are the metrics we used to represent the popularity of repositories on GitHub. Columns 2 through 6 from the left represent various information about a particular linear model, ranging from independent variables for a particular model (column 2 from left) to p-values and their significance marks (columns 5 and 6 respectively) for different independent variables in a model. The last four columns on the right (column 7 through 10) represent various information about the Analysis of Variance for the different model variables. A different model was built for each of the three dependent variables, and so the results in Table III are for three different models. We next present the results for each of the three models.

1) Relationship between code review in DHIS2 repositories and the number of GitHub forks: With reference to Table III, the model in which the dependent variable is *Forks* has 0.5346 and 0.5222 as values of Multiple R-squared and Adjusted R-squared respectively, suggesting that it (the model) accounts for 53.5% of the dependent variable variance. This average value of variance in dependent variable could be reflective of the diversity in repository sizes, despite the fact that values of the dependent and independent variables were all extracted from GitHub and all the repositories involved in the present study were owned by the DHIS2 developer community. The *t-values*, *p-values* and *Sum Sq* suggest that only the number of commits and the number of code review comments significantly affected the number of forks amongst DHIS2 repositories on GitHub. This implies that sizeable DHIS2 repositories with reviewed code stand good chances of evoking the interest of developers on GitHub, who might fork and extend the code, with a potential of contributing back. It could also mean that, because repositories with unreviewed (or poorly reviewed) code are prone to more quality issues [9, 10], developers in social coding websites like GitHub tend to shy away from forking and extending them. Future work might conduct deep investigation into this conjecture. However, we found no direct link between the number of instances in which the recommendations of the code review process are implemented and the number of forks for DHIS2 repositories in GitHub.
Additionally, a positive sign on the the coefficient of *Total Comments* suggests that DHIS2 repositories with higher numbers of code review comments are most likely to be forked.

TABLE III
RELATIONSHIP BETWEEN CODE REVIEW AND POPULARITY OF DHIS2 REPOSITORIES ON GITHUB

	Linear Model					**Analysis of Variance**			
Dependent Variable	**Independent Variables**	**Estimate**	**t-value**	**p-value**		**Sum Sq**	**F-value**	**p-value**	
Forks	(Intercept)	-0.3982	-3.550	0.000563	***				
	Commits	0.4000	7.296	4.39e-11	***	14.13	116.561	< 2e-16	***
	Total Comments	0.7463	3.021	0.003118	**	1.389	11.454	0.000981	***
	Effective Instances	-0.5539	-1.331	0.185937		0.215	1.771	0.185937	
	Multiple R-squared: 0.5346, Adjusted R-squared: 0.5222								
Active Forks	(Intercept)	-0.2192	-2.688	0.008275	**				
	Commits	0.2325	5.834	5.24e-08	***	5.282	82.479	4.06e-15	***
	Total Comments	0.6193	3.449	0.000793	***	0.648	10.112	0.001900	**
	Effective Instances	-0.6029	-1.993	0.048698	*	0.254	3.971	0.048700	*
	Multiple R-squared: 0.4608, Adjusted R-squared: 0.4465								
Stars	(Intercept)	-0.4710	-5.210	8.57e-07	***				
	Commits	0.3987	9.022	5.57e-15	***	11.584	147.161	< 2e-16	***
	Total Comments	0.5483	2.754	0.00686	**	0.318	4.038	0.0469	*
	Effective Instances	-0.6570	-1.959	0.0526	.	0.302	3.837	0.0526	.
	Multiple R-squared: 0.5784, Adjusted R-squared: 0.5672								
	Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1								

This is especially possible in situations where code review improves the quality of code, adding to its comprehensibility, extensibility, and maintainability [9, 10].

2) Relationship between code review in DHIS2 repositories and the number of active forks on GitHub: Focusing on the results for the second dependent variable (*Active Forks*) in Table III, we notice that the model has 0.4608 as the value of Multiple R-squared, and 0.4465 as the value of Adjusted R-squared. This means that the model can explain only 46.1% of the dependent variable variance. This relatively low value of Multiple R-squared suggests that, though the values of both independent and dependent variables were taken from one source, GitHub, developers of original repositories and developers who maintained active forks of the original repositories might have little or no interaction. Moreover, similar to *Forks*, increasing the number of commits and the number of code review comments positively impacts the number of *Active Forks*. This could as well be attributed to the fact that good code review rigor produces good quality software that is easy to understand, maintain and extend [9, 10], which could, in turn, motivate open source developers to continue working on the repositories they fork.

However, the negative sign on the coefficient of *Effective Instances* suggests that the decrease in *Effective Instances* is associated with the increase in *Active Forks*. While it calls for further investigation, this result could imply that, sometimes, spending more time implementing the recommendations made during code review could reduce the activity of forks for DHIS2 repositories in GitHub. In GitHub, developers fork a repository, work on its code, and then contribute back to an original repository through *Pull Requests* [20]. Because developers of an original repository reserve the right to accept or reject any Pull Request sent from a forked repository, more rejections could demoralise potential contributors from working on the repositories they fork. Because most contributions to DHIS2 are overseen by a particular main development node, it could be that new Pull Requests do not easily survive the review process. This is understandable given the sensitivity of the DHIS2 software and the scale of its coverage, but more rejections or delayed approvals for Pull Requests could scare away some contributions of good potential to the long-term sustainability of the software [29].

3) Relationship between code review in DHIS2 repositories and the number of stars on GitHub: Based on the results in Table III, when *Stars* was used as the dependent variable, the model produced 0.5784 as the value of Multiple R-squared, and 0.5672 as the value of Adjusted R-squared, implying that the model represents 57.8% of the variance in the dependent variable. This relatively high value of Multiple R-squared, in addition to the diversity of repository sizes, can be linked to the fact that the values of all independent variables and the dependent variable were all extracted from GitHub, and from the repositories owned by the same organisation or network. Furthermore, based on *t-values*, *p-values* and *Sum Sq*, all the three independent variables in the model–*Commits*, *Total Comments*, and *Effective Instances*–significantly influenced the number of stars on DHIS2 GitHub repositories.

On the one hand, the positive signs on the coefficients of *Commits* and *Total Comments* suggest that DHIS2 GitHub repositories with higher numbers of commits and code review comments are likely to attract more stars. On the other hand, the negative sign on the coefficient of *Effective Instances* suggests that DHIS2 repositories in which higher numbers of reviews are effected have a lower number of stars. Two clues could be related to this result. First, as stated before, spending more time effecting code review recommendations can delay code from reaching developers who want it the most, especially in an open source environment where development is more or less like a marathon in which meeting immediate customer requirements is of paramount importance. In such an environment, developers are likely to star repositories that attend their immediate needs, even if they have unreviewed and/or "low quality" code. Second, it could be that effecting reviews can sometimes produce poor-quality code [27], and this can in turn reduce the number of stars that developers give to a repository.

V. Threats to Validity

The following are the threats to the validity of the results in the present study, as well as their mitigation strategies.

A. Internal Validity

We relied on public DHIS2 repositories on GitHub, so that we might have missed other private DHIS2 repositories with reviewed code. However, we observed that DHIS2 repositories on GitHub are frequently updated, and so public repositories stand good chances of being a true reflection of what is actually practiced by DHIS2 developers. Also, focusing on public GitHub repositories enabled us to study DHIS2 code review practices in a natural environment, without developer intervention, which might have biased the results. Nevertheless, a follow up study with DHIS2 developers, accessing even private repositories (if any) could generate more richer insights on DHIS2 code review practices.

B. Construct Validity

The results of the present study can suffer two main construct validity threats. First, the reliance on code authors' comments to measure code review effectiveness (section III-B) might have missed effective instances of code review where developers had decided to silently implement code review recommendations without responding to comments made on their code. But, even so, if developers do not react to comments made on their code, it can be difficult to determine if they saw and worked on the comments, in the first place. This is especially true in a context like the one in the present study where there were no evidence of systematic tracking of code review knowledge base which, among other things, could help researchers and practitioners to easily determine which part of code was reviewed, by who, and whether or not code review recommendations were effected. As such, focusing on developers' engagement with comments made on their code can be a good proxy for measuring whether or not the comments made during code review were effected. Nevertheless, future work can incorporate new commits from a code author when determining if the comments made during code review helped improve the code.

Second, we had limited understanding of the actual DHIS2 code review practices, and so it is possible that we expected comments even on parts of code where DHIS2 developers were, in practice, expected to remain silent. However, even if that was the case, still, it could hardly justify the fact that almost 80% of the repositories of focus in our study did not have a single code review comment (see Table II). Thus, relying on available comments was the best pragmatic alternative to evidence the practice of code review amongst DHIS2 developer community. Nonetheless, in the future, it might be good to have a measure of code review activity that is informed by actual practices of the development team.

C. External Validity

Since we focused on only public DHIS2 projects on GitHub, the findings of our study might be hardly generalisable to all DHIS2 repositories or to other open source systems in GitHub and other social coding websites. To mitigate the effects of this, we analysed 117 DHIS2 repositories (section III-B) of various sizes which, we believe, represent a large proportion of all repositories maintained by the DHIS2 community of developers. Furthermore, since the public repositories we analysed are open to all members of the GitHub community, and are diverse in terms of aspects like size and purpose, they represent a typical Open Source Software ecosystem. However, to corroborate the insights from the present study, in the future, it would be good to conduct a similar study on OSS projects owned and maintained by diverse developer communities.

D. Conclusion Validity

The results in the present study are based on a dataset of 117 public DHIS2 GitHub repositories which were carefully selected based on the inclusion and exclusion criteria presented in section III-B. In addition, all the study variables were subjected to rigorous statistical analysis (section III-C) before attempting to draw conclusions from them. That scrutiny, in our opinion, minimised the threats to conclusion validity even more.

VI. Conclusions

DHIS2 is currently in use in more than 60 countries across the world to collect, analyse and visualise case-based and aggregated routine health data. Most of its use is in the developing world, including in Africa. Given the role it plays in most countries, it is important to study various aspects of its development, adoption, use, and sustainability. Previous studies on DHIS2 have concentrated on political, social, cultural, and organisational factors affecting its adoption and use. However, the software engineering aspects of DHIS2 have not been widely studied.

In this paper, we have analysed 117 DHIS2 repositories on GitHub to gather empirical evidence about the distribution of code reviews in these repositories, as well as the relationship between code review and the popularity of DHIS2 code repositories on GitHub. Results of the study revealed the presence of minimal code review activity in DHIS2 GitHub repositories, as well as minimal evidence of code improvement based on code review. Also, code reviews were found to affect the popularity of DHIS2 repositories on GitHub.

Part of our future work is to investigate the relationship between code reviews and the popularity of repositories using a wide range of code review and popularity metrics, and on repositories owned by different developer communities, to generate insights that could inform software teams when planning to increase popularity and impact of their repositories on social coding websites like GitHub.

References

[1] J. Braa and S. Sahay, "The DHIS2 open source software platform: evolution over time and space," *Global health informatics. Principles of eHealth and mHealth to improve quality of care. The MIT Press*, 2017.

[2] A. A. Bhattacharya, N. Umar, A. Audu, H. Felix, E. Allen, J. R. M. Schellenberg, and T. Marchant, "Quality of routine facility data for monitoring priority maternal and newborn indicators in dhis2: A case study from gombe state, nigeria," *PLOS ONE*, vol. 14, pp. 1–21, 01 2019.

[3] "District Health Information Software (Version 2)," https://www.dhis2.org/about, Accessed: 7^{th} August 2020.

[4] "DHIS2 for Education," https://www.dhis2.org/education, 2020, Accessed: 7^{th} August 2020.

[5] S. Manoj, A. Wijekoon, M. Dharmawardhana, D. Wijesuriya, S. Rodrigo, R. Hewapathirana, P. Siribaddana, T. Gunasekera, and V. Dissanayake, "Implementation of district health information software 2 (dhis2) in sri lanka," *Sri Lanka Journal of Bio-Medical Informatics*, vol. 3, pp. 109–114, 06 2013.

[6] R.-R. Tohouri, I. Asangansi, O. H. Titlestad, and J. Braa, "The change strategy towards an integrated health information infrastructure: Lessons from sierra leone," in *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 2010, pp. 1–10.

[7] E. Nyella and H. Kimaro, "His standardization in developing countries: Use of boundary objects to enable multiple translations." *African Journal of Information Systems*, vol. 8, no. 1, 2016.

[8] S. Sahay, E. Monteiro, and M. Aanestad, "Configurable politics and asymmetric integration: Health e-infrastructures in india," *Journal of the Association for Information Systems*, vol. 10, no. 5, p. 4, 2009.

[9] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. Association for Computing Machinery, 2014, p. 192–201.

[10] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 168–179.

[11] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 171–180.

[12] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. Association for Computing Machinery, 2014, p. 202–211.

[13] M. Paixao, J. Krinke, D. Han, and M. Harman, "Crop: Linking code reviews to source code changes," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018, pp. 46–49.

[14] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 712–721.

[15] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. Association for Computing Machinery, 2013, p. 202–212.

[16] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 141–150.

[17] 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. Association for Computing Machinery, 2014.

[18] 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.

[19] 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," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, pp. 1–30, 09 2017.

[20] J. Businge, M. Openja, D. Kavaler, E. Bainomugisha, F. Khomh, and V. Filkov, "Studying android app popularity by cross-linking github and google play store," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 287–297.

[21] C. D. Hundhausen, A. Agrawal, and P. Agarwal, "Talking about code: Integrating pedagogical code reviews into early computing courses," *ACM Trans. Comput. Educ.*, vol. 13, no. 3, pp. 1 – 14, Aug. 2013.

[22] 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 *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 1039–1050.

[23] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 122–131.

[24] 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*, 2016, pp. 1–10.

[25] J. Businge, M. Openja, S. Nadi, E. Bainomugisha, and T. Berger, "Clone-based variability management in the android ecosystem," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 625–634.

[26] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 171–180.

[27] J. Shimagaki, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi, "A study of the quality-impacting practices of modern code review at sony mobile," in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 212–221.

[28] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 257–268.

[29] D. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, "" was my contribution fairly reviewed?" a framework to study the perception of fairness in modern code reviews," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 523–534.