A Health Index of Open Source Projects Focusing on Pareto Distribution of Developer's Contribution

Hirohisa Aman*, Aji Ery Burhandenny†, Sousuke Amasaki‡, Tomoyuki Yokogawa‡ and Minoru Kawahara*
*Center for Information Technology, Ehime University, Matsuyama, Ehime 790-8577, Japan
†Graduate School of Science and Engineering, Ehime University, Matsuyama, Ehime 790-8577, Japan
‡Faculty of Computer Science and Systems Engineering, Okayama Prefectural University, Soja, Okayama 719–1197, Japan

***Abstract*—Open source software (OSS) products have been broadly utilized for the IT business as well as the personal use in recent years. Software companies can receive much benefit from OSS products in terms of cost to develop and maintain their products. However, there are also risks that products of interest might become no longer being successfully maintained by the OSS developers because a successful maintenance is not obligation of developers. In order to evaluate a project's health from a perspective of a macroscopic trend analysis, this paper focuses on the distribution of the developer's contribution to an OSS project, and analyzes the relationships of distribution features with the quality of products. The empirical analysis with 32 popular OSS projects shows that the length of tail in the Pareto chart can be a health index of an OSS project in terms of the balance between bug fix and feature upgrade.**

I. Introduction

Recently, open source software (OSS) products have become more and more popular, and have been widely used in various areas of information and communication technology. Leading examples include Linux, Apache HTTP, Firefox and PostgreSQL. These days, many OSS products are utilized for not only personal purposes but also for business. According to a survey report on the utilization of OSS in software-related companies [1], 78 percent of the survey respondents said that they run part or all of its operations on OSS products. In addition, 66 percent of respondents also said their software products are built on OSS ones. Thus, many companies are using and focusing on OSS products in their business, and this trend is highly expected to continue.

However, there are also risks associated with the quality control and maintenance of their software products. For example, if a software company had utilized an OSS product for running their commercial service, and encountered a failure caused by a fault in the OSS product, it would be unsure whether the faulty part of the program would be quickly detected and fixed in the OSS project or not; The company's engineers might have to fight the suspicious programs to fix the fault by themselves, even if they have no experience or knowledge in regard to maintaining the faulty part of the programs. In order to minimize such a risk, software companies should adopt the OSS products which are expected to be actively developed and maintained both now and in the future. Thus, there is a need for evaluating the ability of the OSS project to steadily run—the "health level" of the OSS project. While there are some related criteria such as the popularity and the user assessment, they are based on the products rather than the project. Recently, there have been some studies focusing on the project operation, including the contributions of OSS developers [2]. It would be promising to focus on the developers in order to discuss the project's health because their contributions drive the OSS project. While an analysis of individual's contribution must be beneficial, such a detailed analysis tend to be unfit for capturing the entire spectrum of the project. Therefore, in this paper, we will conduct a macroscopic analysis of the developer's contribution to the OSS project, and propose a health index of project.

The key contributions of this paper are the followings.

1) A quantitative analysis of developer's contribution is conducted for 32 popular OSS projects, and reports their features from the perspective of the Pareto principle.
2) The analysis result shows that the length of tail in the Pareto chart can be a useful index for evaluating whether the OSS project has been developed in a healthy balance between bug fix and feature upgrade.

The remainder of this paper is organized as follows: Section II describes available data from OSS projects, and proposes metrics of the developer's contribution. Then, Section III reports on our empirical analysis and discussions. Section IV briefly discusses our related work. Finally, Section V gives the conclusion of this paper and our future work.

II. OSS Project and Developer's Contribution

A repository of an OSS project is usually available on the Internet, so anyone can get the products and see any changes that have been made in the past. While an OSS project may use not only a repository but also a bug tracking system and a mailing list system to manage their bug reports and e-mails exchanged among the developers, the access to those material is sometimes restricted to the developers participated in the project. Thus, in this paper, we will focus on data managed in a repository which anyone can freely access.

We can investigate various data in regard to each commit to an OSS project through the project's repository, e.g., when the file change was made, what the difference from the previous revision was, and who made it. Recently, many studies analyzing repositories have been reported in the Mining Software Repositories (MSR) community [3]: for example, commit log is analyzed to predict fault-prone modules for more effective testings and code review [4], [5], [6].

In this paper, we will analyze the developer's contribution in a macroscopic manner. We begin with quantifying each developer's contribution by using the following two metrics.

1) Number of Commits made by the developer (NoC), and
2) Ratio of Files in which the developer has been involved (RoF):
$$\text{RoF}(d) = \frac{n(d)}{N},$$
where d signifies a developer, N is the number of files included in the latest version of the product, and $n(d)$ is the the number of files which have been added or modified by d and are included in the latest version.

Table I presents an example, where 2 developers A and B added or modified 8 files through 10 commits; each row corresponds to each commit made by the developer shown in the rightmost column, and a check mark (✓) signifies the corresponding file was added or modified at the time. For instance, developer A added 4 files whose file numbers are 1–4 at the commit no.1, and developer B did 4 files whose file numbers are 5–8 at the commit no.2. At the commit no.5, developer B modified 2 files whose numbers are 2 and 6. In this example, we have NoC(A) $= 4$ and NoC(B) $= 6$, because developer A made the commits whose numbers are 1, 3, 7 and 8, and developer B did the remaining 6 commits. For metric RoF, we obtain the following values: $\text{RoF}(A) = 4/8 = 0.5$ and $\text{RoF}(B) = 7/8 = 0.875$, since $N = 8$, $n(A) = 4$ and $n(B) = 7$ as shown in Table I.

The NoC is a simple metric observing the degree of the developer's contribution—the more commits, the higher the NoC value. A highly contributing developer would actively commit files to the repository, and then the developer would have a high value of NoC. The RoF is a metric to show how widely the developer has been involved in the product. If a developer has a high value of RoF, the developer must have contributed to many parts of the product, which means there is a high contribution to the project. In this paper, we try to capture an OSS project's status of development and maintenance through the above metrics, and examine whether they can be useful in evaluating the organization of project.

III. Empirical Analysis

This section conducts an empirical analysis on the above two metrics as health indexes of OSS projects.

A. Research Questions

Our aim is to analyze the usefulness of our two metrics for assessing the project's health. While we proposed two different metrics observing each developer's contribution, we do not aim to evaluate the individual developer by using these metrics. Instead, we will focus on the "distribution" of metric values rather than the individual's metric value. Now we tackle the following two research questions.

RQ1: Can the distribution of metric values show a difference among OSS projects?

RQ2: Can the distribution be noteworthy for making a useful health index of OSS project?

TABLE I
Example of 10 commits involving 8 files by 2 developers.

commit no.	file no.								committing developer
	1	2	3	4	5	6	7	8	
1	✓	✓	✓	✓					A
2					✓	✓	✓	✓	B
3	✓								A
4					✓		✓	✓	B
5		✓				✓			B
6		✓		✓					B
7			✓	✓					A
8	✓	✓							A
9						✓	✓	✓	B
10			✓	✓					B
involved developer	A	A B	A B	A B	B	B	B	B	

RQ1 is a basic question in regard to the meaningfulness of our two metrics. If there is no difference in the metric values among projects, the proposed metrics are not sensible and are not beneficial to our discussion of OSS project's health. RQ2 is a key question concerning the contribution of this paper. We will try to find a feature of the distribution of metric values which would be useful in assessing the project's health.

In this empirical analysis, we use the following bug fix rate (BFR) as a criterion for evaluating a project:
$$\text{BFR} = \frac{\text{number of bug fixing commits}}{\text{number of all commits}}\ .$$
Any OSS project may have not only bug fixing commits but also other commits for enhancements. The balance between the bug fix and the enhancement is a useful measure to assess the project's evolution. It is generally undesirable to have many bug fixing commits during the development. A low value of BFR would signify a successful evolution of the software product, and it means the project has been running healthily. While a project with an extremely low BFR value may also be unhealthy, we assume that projects of interest have many commits for both bug fixes and enhancements, and we mainly focus on their balance through BFR in this paper.

B. Empirical Objects and Data Collection

We collected 60 popular OSS projects shown in Table II. The key reasons why we used them are 1) they are popular projects, and 2) their products have been managed with Git.

Reason 1) is for increasing the value and the generality of our results. Since any finding derived from minor projects must be worthless, we focused only on popular OSS projects. They are ranked in top 100 of popularity at SourceForge.net[1] or widely known projects in the world. Moreover, we will excluded the projects which have been driven by a few developers, from our data set because they may have project-specific trends, i.e., a lack of generality in their data. As we will describe later, 28 out of 60 projects end up being excluded since the number of unique developers is less than 10. Reason 2) is for the ease of data access: we can easily make a clone of repository, so our data collection is quickly performed.

[1]https://sourceforge.net/

TABLE II
OSS projects analyzed in this paper.

no.	project
1	android-x86
2	Angry IP Scanner
3	AutoClicker
4	Catacombae
5	Cool Reader
6	Dev-C++
7	Dia Diagram Editor
8	Ditto
9	Eclipse Checkstyle Plug-in
10	Eclipse Tomcat Plugin
11	eXo Platform
12	Expat XML Parser
13	Ext2 Filesystems Utilities (Ext2 Util)
14	Firebird
15	FlightGear
16	FreeMind
17	Git
18	GNU ARM Eclipse
19	GParted
20	Hibernate ORM
21	HPCC Platform
22	KDiff3
23	libjpeg-turbo
24	Linux
25	LinuxonAndroid
26	MediaPortal
27	MediathekView
28	milter manager
29	Money Manager Ex
30	Moodle
31	Nagios Core
32	NAPS2
33	net-snmp
34	Notepad++ Compare plugin
35	Notepad++ Plugin Manager
36	Notepad++ Python Script
37	Pandora FMS
38	Password Safe
39	PMD
40	PNP4Nagios
41	ProjectLibre
42	PSeInt
43	rEFInd
44	Ruby
45	Ruby on Rails
46	Scribus
47	Smoothwall
48	SQuirreL SQL Client (SQuirreL)
49	SystemRescueCd
50	TEncoder Video Converter
51	The OpenGL Extension Wrangler Library (OpenGL Lib)
52	tuxboot
53	Ubuntuzilla: Mozilla Software Installer
54	UNetbootin
55	Universal Media Server (Universal Media)
56	Win32 Disk Imager
57	Wine
58	Wineskin
59	WordPress
60	xCAT

TABLE III
Numbers of files, developers and commits in each project.

no.	project	#files	#developers	#commits
1	android-x86	49,177	12,121	594,738
2	Angry IP Scanner	769	11	1,034
5	Cool Reader	3,378	21	4,077
11	eXo Platform	1,983	179	11,474
12	Expat XML Parser	194	20	1,394
13	Ext2 Util	1,918	151	5,494
15	FlightGear	913	57	4,650
17	Git	2,941	1,421	55,431
20	Hibernate ORM	10,360	267	7,149
21	HPCC Platform	9,406	52	16,505
23	libjpeg-turbo	433	15	1,629
24	Linux	54,376	13,888	700,908
26	MediaPortal	6,918	42	7,839
29	Money Manager Ex	640	16	3,652
30	Moodle	14,968	597	81,772
31	Nagios Core	866	19	2,262
33	net-snmp	2,198	61	26,144
37	Pandora FMS	4,705	33	12,328
38	Password Safe	2,621	56	10,231
39	PMD	2,123	99	8,698
40	PNP4Nagios	701	29	1,143
44	Ruby	4,364	97	43,977
45	Ruby on Rails	3,149	3,551	60,180
47	Smoothwall	1,506	14	1,548
48	SQuirreL	7,444	31	7,956
51	OpenGL Lib	270	29	1,118
54	UNetbootin	270	10	628
55	Universal Media	1,297	41	6,753
56	Win32 Disk Imager	47	10	159
57	Wine	141,484	3,172	22,707,259
59	WordPress	1,596	71	34,263
60	xCAT	2,150	79	15,760

For each project, we collected data in the following scheme.

1) Get commit data of each developer.
   Collect the commit logs, and classify them according to the developer. If there are two or more developers who seem to be the same one—they have the same name or e-mail address—, merge their data into a single set.
2) Collect metric values of NoC and RoF.
   For each developer, calculate the NoC value and the RoF value.
3) Calculate the BFR value.
   For each project, count all commits and all bug fixing commits, and calculate the BFR value as the ratio of the latter to the former. These numbers can be obtained by counting the unique hash codes which appear in all of the commit logs and in the bug fix commit logs, respectively. The decision as to whether a commit was a bug fix or not is made by checking the appearance of a bug fix-related keyword in the commit message, according to generally accepted practice in the MSR community, proposed by Śliwerski et al. [7].

C. Results and Discussions

Table III shows the total numbers of files, developers and commits in each project, except for the ones whose number of developers is less than 10; 28 out of 60 projects are excluded from our data set due to the small number of developers, so 32 projects are shown in Table III.

We first checked the correlations among three numbers: we calculated Spearman's rank correlation coefficient ρ [8] rather than Pearson's coefficient because of their skewed distributions. They were ρ(#files, #developers)$= 0.718$, ρ(#files, #commits)$= 0.803$ and ρ(#developers, #commits)$= 0.872$. These three numbers seem to be strongly related each other, i.e., we can simply say that the more files, the more developers and the more commits. It is a natural trend, so our data set may provide a general data of OSS projects.

1) Can the distribution of metric values show a difference among OSS projects? (for RQ1): Tables IV and V show the distributions of NoC values and of RoF values, respectively—the quartiles in the set of developers. The minimum values of RoF result in 0% because some files were modified in the past but are not included in the latest version of the product.

In a majority of projects, the maximum values of NoC are over $1,000$, i.e., there are some highly-contributing developers in those projects. However, such developers are a minority in the projects other than WordPress since these medians are less than 100. On the other hand, most of WordPress's developers have committed a few dozen times or more, and the project shows a different distribution than the others.

While there are variations of RoF values among projects, the 75 percentiles of RoF values are less than 10% except for 5 projects (Expat XML Parser, Money Manager Ex, Smoothwall, Win32 Disk Imager and WordPress). That is to say, in many projects, there are a few developers who added or modified files extensively in the directory trees. For example,

TABLE IV
DISTRIBUTIONS OF DEVELOPERS' CONTRIBUTIONS: NoC.

project	min	25%	50%	75%	max
android-x86	1	1	2	11	18,199
Angry IP Scanner	1	1	3	77	466
Cool Reader	1	1	2	7	3,859
eXo Platform	1	3	18	76.5	1,242
Expat XML Parser	1	1	2	28.5	549
Ext2 Util	1	1	2	5	3,923
FlightGear	1	2	4	100	969
Git	1	1	2	7	16,671
Hibernate ORM	1	1	1	5	2,357
HPCC Platform	1	1.5	45	157	7,658
libjpeg-turbo	1	2	2	9	1,552
Linux	1	1	3	12	20,876
MediaPortal	1	2	8	29	3,834
Money Manager Ex	1	6	51	241	1,315
Moodle	1	1	5	37	7,948
Nagios Core	1	1.5	5	10.5	974
net-snmp	1	1	2	23	7,818
Pandora FMS	1	34	79	406	2,651
Password Safe	1	1.5	6	33.5	5,334
PMD	1	1	3	27	3,352
PNP4Nagios	1	1	3	8	922
Ruby	1	21	83	294	13,090
Ruby on Rails	1	1	1	3	4,423
Smoothwall	1	6	46.5	176	534
SQuirreL	1	1	4	24	3,553
OpenGL Lib	1	1	2	6	425
UNetbootin	1	1	1.5	2	606
Universal Media	1	1	5	20	4,355
Win32 Disk Imager	1	2	5.5	25	76
Wine	1	1	4	34	108,116
WordPress	1	31.5	156	432.5	7,153
xCAT	1	11.5	25	141	2,754

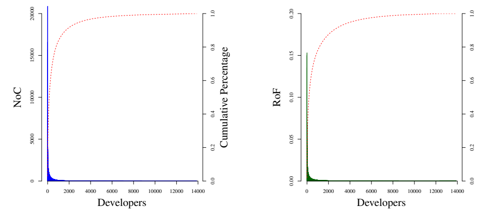
TABLE V
DISTRIBUTIONS OF DEVELOPERS' CONTRIBUTIONS: RoF (%).

project	min	25%	50%	75%	max
android-x86	0	0	0	0	16.9
Angry IP Scanner	0	0.1	0.5	8.8	71.5
Cool Reader	0	0	0.1	0.3	98
eXo Platform	0	0	0.1	1.3	43.6
Expat XML Parser	0	0.5	1	12.9	60.3
Ext2 Util	0	0.1	0.1	0.4	69.3
FlightGear	0	0.1	0.5	4.7	40.5
Git	0	0	0.1	0.2	50.4
Hibernate ORM	0	0	0	0.1	85
HPCC Platform	0	0	0.5	2.8	70.1
libjpeg-turbo	0	0.2	0.9	5.3	99.3
Linux	0	0	0	0	15.3
MediaPortal	0	0	0.1	0.7	71.1
Money Manager Ex	0	0.6	8.1	30.9	92.3
Moodle	0	0	0	0.2	34.6
Nagios Core	0	0.2	0.4	2.9	72.9
net-snmp	0	0	0.1	1.4	96
Pandora FMS	0	0.4	1.7	6.8	38.5
Password Safe	0	0	0	0.6	93.3
PMD	0	0	0	0.1	95.5
PNP4Nagios	0	0.1	0.4	1	99.1
Ruby	0	0.3	1.2	3.8	60.9
Ruby on Rails	0	0	0.1	0.1	25.8
Smoothwall	0	0.4	2.7	14.1	48.9
SQuirreL	0	0	0	0.5	67.9
OpenGL Lib	0	0.4	0.7	4.4	89.6
UNetbootin	0	0.4	0.4	9.1	100
Universal Media	0	0.1	0.2	0.5	81.9
Win32 Disk Imager	0	5.3	12.8	28.7	70.2
Wine	0	0	0	0	40.8
WordPress	0	1.1	4.8	15.4	52.1
xCAT	0	0.3	0.9	3.4	28.2

the maximum RoF value in Linux is 15.3%; Since Linux is an operating system and provides a wide range of functions, it is harder to comprehensively contribute the development of all files than the other products. Notice that we discuss the contribution "as the author" in this paper, so we do not deny that there are "reviewers" checking a wide range of the files comprehensively. As is the case in NoC, WordPress project shows a different distribution of RoF values from the others.

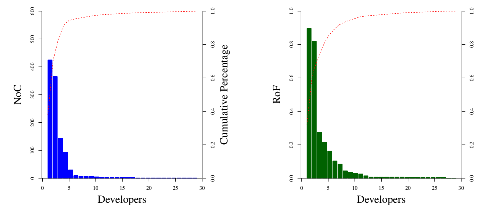
To quantitatively compare these distributions, we introduce the notion of the Pareto chart [9]. Figures 1, 2 and 3 show the Pareto charts of NoC values and RoF values in Linux, OpenGL Lib and WordPress, respectively; we show only three projects' Pareto charts for lack of space. The vertical axes signify the degree of the developer's contribution (the left hand side) and the cumulative percentage (the right hand side), and the horizontal axis corresponds to developers in decreasing order of contribution.

Since a minority of developers look making most of the total contributions, the Pareto principle seems to roughly hold for most projects. The Pareto principle says that 80% of the total contributions are made by 20% of the developers where the percentages may vary, and the Pareto chart has a "long tail" according to the remaining 80% of the developers. We checked the rate of the dominant developers who made 80% of the total contributions, and calculated the relative length of the tail (see Table VI): WordPress required more developers to make 80% of the total contributions than the other projects. That is to say, WordPress had a shorter tail than the others.



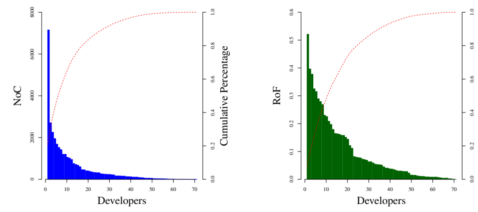
(a) NoC (b) RoF

Fig. 1. Pareto charts of NoC values and RoF values in Linux.



(a) NoC (b) RoF

Fig. 2. Pareto charts of NoC values and RoF values in OpenGL Lib.



(a) NoC (b) RoF

Fig. 3. Pareto charts of NoC values and RoF values in WordPress.

TABLE VI
PERCENTAGES OF DOMINANT DEVELOPERS AND THE OTHERS

	NoC		RoF	
project	dominant	other(tail)	dominant	other(tail)
android-x86	5.6	94.4	7.8	92.2
Angry IP Scanner	9.1	90.9	8.3	91.7
Cool Reader	4.8	95.2	4.5	95.5
eXo Platform	24.6	75.4	12.2	87.8
Expat XML Parser	10.0	90.0	19.0	81.0
Ext2 Util	1.3	98.7	5.3	94.7
FlightGear	17.5	82.5	19.0	81.0
Git	2.6	97.4	12.1	87.9
Hibernate ORM	4.1	95.9	2.6	97.4
HPCC Platform	9.6	90.4	11.3	88.7
libjpeg-turbo	6.7	93.3	12.5	87.5
Linux	5.6	94.4	8.3	91.7
MediaPortal	4.8	95.2	4.7	95.3
Money Manager Ex	18.8	81.3	29.4	70.6
Moodle	6.9	93.1	7.2	92.8
Nagios Core	5.3	94.7	10.0	90.0
net-snmp	8.2	91.8	9.7	90.3
Pandora FMS	21.2	78.8	23.5	76.5
Password Safe	3.6	96.4	5.3	94.7
PMD	6.1	93.9	2.0	98.0
PNP4Nagios	3.4	96.6	3.3	96.7
Ruby	14.4	85.6	21.4	78.6
Ruby on Rails	2.6	97.4	11.7	88.3
Smoothwall	21.4	78.6	20.0	80.0
SQuirreL	6.5	93.5	6.3	93.8
OpenGL Lib	6.9	93.1	13.3	86.7
UNetbootin	10.0	90.0	9.1	90.9
Universal Media	4.9	95.1	7.1	92.9
Win32 Disk Imager	20.0	80.0	36.4	63.6
Wine	5.9	94.1	6.2	93.8
WordPress	23.9	76.1	33.3	66.7
xCAT	17.7	82.3	26.3	73.8

The answer to RQ1 results in "Yes": while the distributions seems to roughly obey the Pareto principle, there are differences in the relative length of tail among projects. The length of tail in the NoC value ranges from 75.4% to 98.7%, and that in the RoF value ranges from 63.6% to 98%, respectively.

2) Can the distribution be noteworthy for making a useful health index of OSS project? (for RQ2): Table VII shows BFR values in each project. WordPress project had the maximum BFR, therefore we can expect that there is a correlation between the length of tail and the BFR. We calculated Pearson's correlation coefficient for each pair and did a t-test whether the correlation is statistically significant ($\neq 0$) or not. They were

- r(length of tail in NoC, BFR)$= -0.244$
  where $t = -1.3766$, $df = 30$ and $p = 0.1788$;
- r(length of tail in RoF, BFR)$= -0.455$
  where $t = -2.7975$, $df = 30$ and $p = 0.0089$.

While the length of tail in NoC values seems to have no correlation with BFR values, the length of tail in RoF values would have a negative correlations with BFR values. That is to say, the longer tail in RoF, the lower BFR.

According to the notion of the Pareto principle, we consider 80% to be a baseline of the length of tail, then we compare BFR values between two groups of projects—the projects whose relative length of tail in RoF is shorter than 80% versus the ones whose relative length of tail in RoF is longer than or equal to 80%. Figure 4 shows the results. From the figure, we

TABLE VII
BFR VALUES OF SURVEYED OSS PROJECTS.

project	BFR	project	BFR
android-x86	0.300	net-snmp	0.192
Angry IP Scanner	0.082	Pandora FMS	0.379
Cool Reader	0.391	Password Safe	0.064
eXo Platform	0.025	PMD	0.021
Expat XML Parser	0.220	PNP4Nagios	0.144
Ext2 Util	0.413	Ruby	0.238
FlightGear	0.208	Ruby on Rails	0.136
Git	0.190	Smoothwall	0.181
Hibernate ORM	0.082	SQuirreL	0.052
HPCC Platform	0.229	OpenGL Lib	0.132
libjpeg-turbo	0.175	UNetbootin	0.234
Linux	0.299	Universal Media	0.157
MediaPortal	0.111	Win32 Disk Imager	0.289
Money Manager Ex	0.206	Wine	0.009
Moodle	0.164	WordPress	0.536
Nagios Core	0.409	xCAT	0.271

clearly see the projects with longer tails in RoF values tend to have lower BFR values. Since the RoF of a developer is the ratio of files which the developer has been involved in, a healthier project may be a project such that various functions have been developed by various developers cooperatively.

While many OSS projects have been successfully run thanks to talented developers who have made a lot of contributions, the workload balance must be a significant factor to assess the project's health. Since it is not sure whether a highly-contributing developer will continuously provide their help to the project in the future, a high workload on a certain developer can be a threat to the project's health. Focusing on the structure of developers in the OSS project would be useful in understanding and evaluating the project.

The answer to RQ2 results in "Yes": the relative length of the tail in the RoF Pareto chart has a negative correlation with the project's BFR, so it can be one of useful health indexes.

D. Threats to Validity

Since our BFR data is based on a simple keyword matching, it might overlook some bug fix commits. However, such a method has been widely used in the MSR community and has a certain level of validity, therefore the data collection scheme would not be a serious threat to validity of our empirical study.

Although we used only 32 projects, they are popular ones from different domains and have been developed by 10 or more developers. Moreover, their programming languages are not the same. Thus, while our data selection was not a random sampling, it would not become a significant threat to validity.

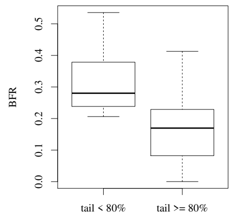


Fig. 4. Comparison of BFR values: OSS projects whose tail length of RoF is shorter than 80% vs. longer than or equal to 80%.

We focused only on the number of commits and not on what kind of changes were made. While impacts of changes should also be considered, we missed those analyses in this paper. Moreover, although we checked only commit "authors," there are other ways of contributing OSS projects: for example, posting a bug report to the bug tracking system or to the support mailing list. Such another kind of contribution would also be important factor for evaluating the OSS project's health level. Addressing these issues are our significant future work.

IV. Related Work

Raja and Tretter [10] focused on the survivability of OSS projects, and they proposed "viability index" which is a linear combination of three sub indexes on 1) vigor, 2) resilience and 3) organization: 1) the vigor index is a measure of project growth; 2) the resilience index means an ability to react to the changes in the project's environment; 3) the organization index is a measure of the diversity of interactions among project members. While the viability index can produce remarkable evaluations of OSS projects, it requires to find appropriate coefficients to combine the above three factors in their model. Although our approach is simpler and less informative than Raja and Tretter model, our one does not need any parameter tuning but it captures yet another feature of projects, so that it would be a useful supplement in evaluating OSS projects.

Yamashita et al. [11] investigated many OSS projects from the perspective of the Pareto principle, i.e., whether the core developers's work dominates the project or not. Their work is an important replication study of the prior work [12], [13], [14], [15], [16], [17], by conducting a large-scale data collection from the GitHub. They showed that a majority of projects are not compliant with the Pareto principle in terms of the core developers's contribution to projects. That is to say, there seems to be a wide variance in the proportion of core developers to non-core ones among projects. While their results look difference from our results, their work included the projects driven by a few developers as well. In our analysis, we excluded such projects and then focused on the length of tail in the Pareto chart. If we include the projects driven by small number of developers (≤ 10), we cannot say that most projects seem to obey the Pareto principle. Notice that our main focus is to capture a feature of OSS project by using the length of tail in the Pareto chart. Moreover, we analyzed the relationships with the bug fix rates in OSS projects as well.

V. Conclusion and Future Work

In this paper, we focused on developers' contributions to OSS projects, and quantified them with using two metrics, the number of commits (NoC) and the ratio of files which the developer has added or modified (RoF). Then, we examined 32 popular OSS projects which have been driven by 10 or more developers, and showed the distribution of the developer's contribution roughly obeyed the Pareto principle. That is to say, the developers could be categorized into two types: the dominant developers (making 80% of the total contributions) and the others. The latter developers form a long tail in the Pareto chart, and the length of the tail in the chart showed an interesting trend: the longer tail in the Pareto chart of RoF, the lower rate of bug fixes in the project. Thus, it would be a useful index to assess the project's health.

To examine the usefulness of proposed index in predicting project's healthiness, we have to perform further investigations with other OSS projects which are not only well maintained but also no longer maintained in the future: we plan to collect a lot of projects from GitHub and analyze them. Moreover, a further analysis focusing on not only the commits but also the pull requests is also our important future work.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments on the first version of this paper. This work was supported by JSPS KAKENHI #16K00099.

References

[1] Black Duck Software, "The ninth annual future of open source survey | black duck," https://www.blackducksoftware.jp/future-of-open-source, Apr. 2015.

[2] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *Proc. Int'l Conf. Softw. Eng.*, May 2013, pp. 452–461.

[3] A. E. Hassan, "The road ahead for mining software repositories," in *Proc. Frontiers of Softw. Maintenance*, 2008, pp. 48–57.

[4] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *Proc. 21st IEEE Int'l Conf. Softw. Maintenance*, 2005, pp. 263–272.

[5] Y. Kamei, S. Matsumoto, A. Monden, K. ichi Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *Proc. 2010 IEEE Int'l Conf. Softw. Maintenance*, 2010, pp. 1–10.

[6] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu, "Bugcache for inspections : Hit or miss?" in *Proc. 19th ACM SIGSOFT Symp. & 13th European Conf. Foundations of Softw. Eng.*, 2011, pp. 322–331.

[7] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *Proc. Int'l Workshop on Mining Softw. Repositories*, May 2005, pp. 1–5.

[8] S. Siegel, *Nonparametric statistics for the behavioral sciences*. McGraw-Hill, 1956.

[9] J. M. Juran and F. M. Gryna, Eds., *Juran's Quality Control Handbook*. New York: McGraw-Hill, 1988.

[10] U. Raja and M. J. Tretter, "Defining and evaluating a measure of open source project survivability," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 163–174, Jan 2012.

[11] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi, "Revisiting the applicability of the pareto principle to core development teams in open source software projects," in *Proc. 14th Int'l Workshop on the Principles of Softw. Evolution*, Aug. 2015, pp. 46–55.

[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, July 2002.

[13] T. T. Dinh-Trong and J. M. Bieman, "The FreeBSD project: a replication case study of open source development," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 481–494, June 2005.

[14] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," *Inf. Syst. J.*, vol. 12, no. 1, pp. 27–42, Jan. 2002.

[15] M. Goeminne and T. Mens, "Evidence for the pareto principle in open source software activity," in *Joint Proc. 1st Int'l Workshop Model Driven Softw. Maintenance & 5th Int'l Workshop Soft. Quality & Maintainability*, Mar. 2011, pp. 74–82.

[16] G. Robles, S. Koch, J. M. González-Barahona, and J. Carlos, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool," in *Proc. 2nd ICSE Workshop Remote Analysis & Measurement of Softw. Syst.*, May 2004, pp. 51–55.

[17] J. Geldenhuys, "Finding the core developers," in *Proc. 36th Euromicro Conf. Softw. Eng. & Advanced Applications*, Sept. 2010, pp. 447–450.