Post Release Versions based Code Change Quality Metrics

Meera Sharma
Swami Shraddhanand College
University of Delhi
Delhi, India
91-9891343835
meerakaushik@gmail.com

Madhu Kumari
Department of Computer Science
University of Delhi
Delhi, India
91-9899324594
mesra.madhu@gmail.com

V B Singh
Delhi College of Arts & Commerce
University of Delhi
Delhi, India
91-9911351168
vbsinghdcacdu@gmail.com

ABSTRACT
Software Metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute. Bug fixing, new features (NFs) introduction and feature improvements (IMPs) are the key factors in deciding the next version of software. For fixing an issue (bug/new feature/feature improvement), a lot of changes have to be incorporated into the source code of the software. These code changes need to be understood by software engineers and managers when performing their daily development and maintenance tasks. In this paper, we have proposed four new metrics namely code change quality, code change density, file change quality and file change density to understand the quality of code changes across the different versions of five open source software products, namely Avro, Pig, Hive, jUDDI and Whirr of Apache project. Results show that all the products get better code change quality over a period of time. We have also observed that all the five products follow the similar code change trend.

Categories and Subject Descriptors
Software and its engineering~Maintaining software

Keywords
Open Source Software, Entropy, New feature, Feature improvement, Software Repositories

1. INTRODUCTION
Open source software undergoes continuous changes, through which new features are added, existing features are improved, bugs are fixed, and performance is improved. For fixing a bug, new feature introduction and feature improvement, a lot of changes have to be incorporated into the source code of the software. As a result, the source code complexity increases during the maintenance activity.

WCI '15, August 10 - 13, 2015, Kochi, India
 ISBN 978-1-4503-3361-0/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2791405.2791466

Source code changes histories/version histories provide abundant resources for understanding the evolution of software systems.

These code changes usually need to be understood by software engineers when performing their daily development and maintenance tasks [16]. Previous research proved that understanding code changes is the basis of various advanced development tasks, such as troubleshooting unexpected behavior [17] and monitoring maintenance of code clones [18]. It is evident that for fixing different issues (bugs/new features introduction/feature improvements) different files of the software need to be changed.

The source code changes have been quantified using entropy based measures and termed as the complexity of code changes [1 and 2]. The quantification of code changes is based upon the Shannon entropy [3].

Prediction models have been explored for whether a source file will be affected by a certain type of code changes. Fine-grained source code changes (SCC) capture such detailed code changes and their semantics on a statement level [15]. These predictions are computed on the static source code dependency graph and use social network centrality measures and object-oriented metrics. The results show that Neural Network models can predict categories of SCC types. The prediction models can output a list of the potentially change-prone files ranked according to their change-proneness, overall and per change type category.

In this paper, we have proposed four new metrics, namely code change quality, code change density, file change quality and file change density to understand the quality of code changes across the different versions of five open source software products namely Avro, Pig, Hive, jUDDI and Whirr of Apache project

The rest of the paper is organized as follows: Section 2 gives the brief of the work already done based on source code changes. Section 3 discusses the data collection. Section 4 gives some statistical measures of data sets used. Section 5 explains the proposed code change metrics. The results and discussion have been presented in section 6. Section 7 discusses about the threats to validity and finally the paper is concluded in section 8 with future research directions.

2. CODE CHANGES BASED PREDICTION
In literature source code changes have been used for different types of predictions. An approach has been proposed [6] that applies data mining techniques to determine change patterns-sets

of files that were changed together frequently in the past-from the change history of the code base. The hypothesis was that the change patterns can be used to recommend potentially relevant source code to a developer performing a modification task. Results showed that the approach can reveal valuable dependencies by applying the approach to the Eclipse and Mozilla open source projects and by evaluating the predictability and interestingness of the recommendations produced for actual modification tasks on these systems. Another approach [7] based on change and bug histories that compute the co-changes – What artifacts (such as files, classes, methods, and lines) should also change? These approaches have the advantage that they can also identify non-source code artifacts that are frequently modified for a given change

In studies, based on source code changes, textual or syntactic approaches have been widely used. Textual analysis treats the source code as just a piece of text and builds change rules according to the textual properties of changes. For example, [8] analyze how many lines added or deleted in a change and associate change size with faults. Syntactic analysis uses syntactic entities, such as classes/methods/fields. A study [9] analyzed the impact of source code changed by slicing the variable def-use pairs. The Data-flow and program slicing have been combined to show data dependencies. The study suggested that, like change itself, change impact is also a high priority indicator in fault prediction, especially for changes of large scales.

An approach [10] has used software history data mining to find patterns in bug fix changes, thereby automatically categorizing bugs. It is now possible to automatically classify bugs into specific bug types, avoiding the traditional problems of human bug categorization. An approach [11] showed that mining previous bug fixes can produce knowledge about why bugs happen and how they are fixed. In this approach, authors mined the change history of 717 open source projects to extract bug-fix patterns and found that missing null checks and missing initializations were very recurrent and can be automatically detected and fixed.

Reference [12] has explored the advantage of using fine-grained source code changes (SCC) for bug prediction. The study presented a series of experiments using different machine learning algorithms with a dataset from the Eclipse platform to empirically evaluate the performance of SCC and Lines Modified (LM). The results showed that SCC outperforms LM for learning bug prediction models.

The first algorithm [13] that identifies previously unknown frequent code change patterns from a fine-grained sequence of code changes. CODINGTRACKER effectively handled the challenges that distinguish continuous code change pattern mining from the existing data mining techniques.

A new technique for predicting latent software bugs, called change classification is proposed by [14]. Change classification uses a machine learning classifier to determine whether a new software change is more similar to prior buggy changes or clean changes. The features extracted from the revision history of a software project stored in its software configuration management repository are used to train the classifier. The trained classifier can classify changes as buggy or clean, with a 78 percent accuracy and a 60 percent buggy change recall on average.

Recently, an attempt has been made to find the quantitative importance of the complexity of code changes, fixing of bugs, new features introduction and feature improvements and their relationship [19]. The authors have proposed Cobb-Douglas production function based two dimensional and three dimensional diffusion models for the prediction of the potential of complexity of code changes.

In this paper, our objective is to develop code change metrics to predict about the code change quality, maintainability and release readiness.

3. DATA COLLECTION

We have collected data from 5 products, namely Avro, Pig, Hive, jUDDI and Whirr of Apache open source project [4]. The historical code change data has been extracted using Github tool [5]. For every issue to get it fixed, the changes have been made in the source code of the software. The changes have been made in different files of the project. These changes have been quantified using entropy based measures and termed as the complexity of code changes [1 and 2]. The quantification of code changes is based upon the Shannon entropy [3]. The following explains the process to calculate entropy (complexity of code changes) release wise against the fixed issues:

$$H_n(P) = -\sum_{k=1}^{n} (p_k * \log_2 p_k) \quad \text{Where, } p_k \geq 0$$

$$\sum_{k=1}^{n} p_k = 1$$

Where, p_k is the probability of change occurrence and defined as the ratio of number of times k^{th} file changed during a period and the total number of changes for all files in that period.

The entropy, H, depends on the number of files in a software system, as it depends on n. For many software systems, there exist files that are rarely modified. To prevent these files from reducing the entropy measure, instead of dividing by the actual current number of files in the software system, we divide by the number of recently modified files. These results in Normalized Static Entropy, *H (P)* defined as [1]

$$H(P) = \frac{1}{Max\ Entropy\ for\ Distribution} * H_n(P)$$

$$= \frac{1}{\log_2 n} * H_n(P) = -\frac{1}{\log_2 n} * \sum_{k=1}^{n} (p_k * \log_2 p_k)$$

$$= -\sum_{k=1}^{n} (p_k * \log_n p_k)$$

$$\text{where } p_k \geq 0, \forall k \notin 1, 2, \ldots\ldots, n \text{ and } \sum_{k=1}^{n} p_k = 1.$$

Table 1 to 5 show the data of five products, Avro (from July 2009 to July 2014), Pig (from April 2009 to October 2013), Hive (from April 2009 to April 2014), jUDDI (from February 2009 to February 2014) and Whirr (from September 2010 to April 2013). We have calculated entropy for each product by using Normalized Static Entropy.

Apache Avro is a data serialization system. It provides: rich data structures, a container file to store persistent data, Remote procedure call (RPC) and simple integration with dynamic languages.

Table 1. Avro Dataset

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
1.0.0	261	3.89	17	23	6
1.1.0	79	1.95	15	9	5
1.2.0	57	0.97	15	8	4
1.3.0	360	3.82	71	106	28
1.3.1	35	0.95	6	10	2
1.3.2	35	0.95	5	4	1
1.3.3	93	2.90	13	11	0
1.4.0	107	2.90	33	21	17
1.4.1	28	0.99	5	5	2
1.5.0	272	4.85	42	28	13
1.5.1	24	1.99	5	13	5
1.5.2	76	2.95	12	6	2
1.5.3	76	2.95	0	2	0
1.5.4	33	0.99	1	0	1
1.6.0	59	1.97	36	35	12
1.6.1	59	1.97	3	1	1
1.6.2	25	2.95	23	19	2
1.6.3	0	1.00	12	1	1
1.7.0	54	2.92	32	11	5
1.7.1	36	1.00	10	3	3
1.7.2	49	1.97	21	7	1
1.7.3	45	2.98	15	7	3
1.7.4	25	1.95	18	12	2
1.7.5	80	5.89	20	14	8
1.7.6	60	4.97	22	13	8
1.7.7	79	5.92	25	8	5

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

Table 2. Pig Dataset

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
0.2.0	398	1.98	210	36	23
0.3.0	190	1.96	30	8	9
0.4.0	254	2.94	44	16	5
0.5.0	265	0.97	6	1	2
0.6.0	809	4.88	100	29	17
0.7.0	287	1.96	150	42	14
0.8.0	744	6.81	188	68	28
0.8.1	458	3.88	41	6	0
0.9.0	271	2.92	204	48	14
0.9.1	133	2.97	28	5	2
0.9.2	153	2.94	35	8	1
0.10.0	107	2.94	161	57	16
0.10.1	684	8.86	41	3	3
0.11	27	0.94	208	88	22
0.11.1	64	1.98	14	2	1
0.12.0	452	5.89	184	62	21

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

Table 3. Hive Dataset

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
0.3.0	597	7.80	144	20	24
0.4.0	887	5.79	122	49	36
0.4.1	137	1.92	10	0	0
0.5.0	1038	1.96	108	37	37
0.6.0	634	7.73	97	52	36
0.7.0	413	4.78	143	88	38
0.7.1	199	2.89	5	1	0
0.8.0	511	5.82	184	109	33
0.8.1	68	1.92	8	2	2
0.9.0	158	1.92	72	52	17
0.10.0	897	8.65	238	54	27
0.11.0	843	3.91	257	59	26
0.12.0	1854	4.83	269	64	17
0.13.0	1671	5.76	519	132	27
0.13.1	330	1.95	22	0	0
0.14.0	107	0.96	698	109	26

Apache jUDDI is an open source Java implementation of Advancing Open Standards for the Information Society (OASIS) the Universal Description, Discovery, and Integration (UDDI) specification for (Web) Services.

Table 4. jUDDI Dataset

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
0.9rc1	905	5.91	0	0	0
0.9rc2	905	5.91	0	0	0
0.9rc3	10	1.00	0	0	0
0.9rc4	442	4.91	1	0	0
2.0rc5	460	5.77	40	0	0
2.0rc6	717	6.85	0	0	1
3.0alpha	717	6.85	8	4	2
2.0rc7	138	2.79	0	0	0
3.0beta	173	1.00	16	1	1

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
2.0	108	1.98	3	1	0
2.0.1	108	1.98	1	0	0
3.0rc1	446	1.90	13	3	0
3.0	446	1.90	12	0	0
3.0.1	96	3.91	34	0	0
3.0.2	40	2.90	28	0	0
3.0.3	50	1.96	7	0	1
3.0.4	20	2.99	2	2	1
3.1.0	224	5.84	44	8	1
3.1.1	33	0.97	12	1	1
3.1.2	0	0.00	4	1	0
3.1.3	14	3.00	3	0	0
3.1.4	10	0.98	6	0	2
3.1.5	300	2.90	20	8	4
3.2	513	8.64	85	32	22

Apache Whirr is a set of libraries for running cloud services. It provides: a cloud-neutral way to run services, a common service API and smart defaults for services. It can be used as a command line tool for deploying clusters.

Table 5. Whirr Dataset

Version	No. of Changed Files	Entropy	Bugs	IMPs	NFs
0.1.0	73	3.85	9	20	6
0.2.0	28	1.92	12	10	2
0.3.0	69	1.88	21	22	3
0.4.0	52	1.92	14	15	4
0.5.0	138	2.93	8	21	5
0.6.0	28	1.94	7	15	5
0.7.0	95	3.89	14	28	5
0.7.1	72	1.95	9	7	0
0.8.0	122	4.78	34	30	4
0.8.1	26	2.00	6	5	0
0.8.2	12	2.97	8	11	4

4. STATISTICAL MEASURES

We have calculated some statistical measures for data sets of all the products. Table 6 shows Minimum (Min), Maximum (Max), Average (Avg.) and Standard deviation (Std.Dev.) for time gap between two versions, number of files changed, number of issues (bugs/improvements/new features) fixed in a version.

Table 6. Statistical Measures

Statistical Measures		Apache Products				
		Avro	Pig	Hive	jUDDI	Whirr
Time gap between versions (months)	Min	0.00	1.00	2.00	0.00	2.00
	Max	6.00	9.00	9.00	30.00	10.00
	Avg.	2.19	3.37	4.18	4.83	2.50
	Std. Dev.	1.75	2.26	2.29	6.67	1.66
Files changed	Min	0	27	68	0	12
	Max	360	809	1854	905	138
	Avg.	81.04	331.00	646.50	286.46	65.00
	Std. Dev.	84.87	242.13	536.66	294.33	41.04
Bugs	Min	0	6	5	0	6
	Max	71	210	698	85	34
	Avg.	19.08	102.75	181.00	14.13	12.91
	Std. Dev.	15.22	80.05	190.49	20.02	8.22
IMPs	Min	0	1	0	0	5
	Max	106	88	132	32	30
	Avg.	15.08	29.94	51.75	2.54	16.73
	Std. Dev.	20.73	28.03	41.76	6.69	8.22
NFs	Min	0	0	0	0	0
	Max	28	28	38	22	6
	Avg.	5.71	11.87	21.63	1.50	3.45
	Std. Dev.	6.35	9.21	14.09	4.47	2.02

We observed that average time gap between two versions is maximum 4.83 months for jUDDI. For Hive product, we have maximum value of average number of files changed and maximum value for average number of issues (bugs/improvements/new features) fixed in a version.

5. CODE CHANGE METRICS

In this section, we have proposed four metrics to calculate the source code change quality and source code change density in terms of code and files changed during the maintenance activities (bugs fixing, new features addition and feature improvements) of open source software. Using these metrics we can quantify the quality of code changed and files changed during maintenance activities of open source software. We defined Code Change Quality as

$$Code\,Change\,Quality\,(CCQ) = \frac{Entropy\,of\,n^{th}\,Version}{Bugs\,of\,(n+1)^{th}\,Version}$$

We have quantified the code change quality with the intuition that current changes in the source code will lead to occurrence of bugs in the future versions [1 and 2]. A high value of CCQ indicates better code change quality of n^{th} version as it produces less bugs in future version.

We defined Code Change Density as

$$Code\,Change\,Density\,(CCD) = \frac{Entropy\,of\,n^{th}\,Version}{Issues\,fixed\,of\,n^{th}\,Version}$$

This gives the quantified value of code change per unit issue (bug/new feature/improvement) fixed in current version. Low value for this metric shows less change in the source code of the software to fix the issues.

We can also quantify the change quality in terms of files changed in current version. For this we defined File Change Quality as

$$File\ Change\ Quality\ (FCQ) = \frac{No.\ of\ Files\ changed\ in\ n^{th}\ Version}{Bugs\ of\ (n+1)^{th}\ Version}$$

Changes in more files will result in complex and buggy code. This shows that the low value of FCQ will lead to less number of bugs in future version.

We defined File Change Density as

$$File\ Change\ Density\ (FCD) = \frac{No.\ of\ Files\ changed\ in\ n^{th}\ Version}{Issues\ of\ n^{th}\ Version}$$

This gives the quantified file change per unit issue fixed in current version. Low value for this metric show less change in code hence less bugs will occur in future.

6. RESULTS AND DISCUSSION

We have calculated the above defined metrics for data of five products of Apache open source software to analyze the quality of source code change. In the following tables, we have shown the numerical values of different metrics. "a" shows that the value cannot be calculated as we do not have bugs for next version for the current changes in the source code.

Table 7. Avro

Version	CCQ	CCD	FCQ	FCD
1.0.0	0.26	0.08	17.40	5.67
1.1.0	0.13	0.07	5.27	2.72
1.2.0	0.01	0.04	0.80	2.11
1.3.0	0.64	0.02	60.00	1.76
1.3.1	0.19	0.05	7.00	1.94
1.3.2	0.07	0.10	2.69	3.50
1.3.3	0.09	0.12	2.82	3.88
1.4.0	0.58	0.04	21.40	1.51
1.4.1	0.02	0.08	0.67	2.33
1.5.0	0.97	0.06	54.40	3.28
1.5.1	0.17	0.09	2.00	1.04
1.5.2	0.00	0.15	0.00	3.80
1.5.3	2.95	1.48	76.00	38.00
1.5.4	0.03	0.49	0.92	16.50
1.6.0	0.66	0.02	19.67	0.71
1.6.1	0.09	0.39	2.57	11.80
1.6.2	0.25	0.07	2.08	0.57
1.6.3	0.03	0.07	0.00	0.00
1.7.0	0.29	0.06	5.40	1.13
1.7.1	0.05	0.06	1.71	2.25
1.7.2	0.13	0.07	3.27	1.69
1.7.3	0.17	0.12	2.50	1.80
1.7.4	0.10	0.06	1.25	0.78
1.7.5	0.27	0.14	3.64	1.90
1.7.6	0.20	0.12	2.40	1.40
1.7.7	a	0.16	a	2.08

From table 7, we observe that the value of CCQ is less than 1 for all versions except Version 1.5.3 for which it is 2.95. This shows that the change quality in terms of code for this product is uniform throughout the different versions. Low values of this metric show that the quality of code change is good and will lead to less occurrence of bugs in future versions. Similarly, we observe that the value of CCD is less than 1 for all versions except Version 1.5.3 for which it is 1.48. This shows that the code change per unit issue fixed (current version) for each version of this product is very low, which shows that the code change quality is better during the maintenance activities. Values of FCQ and FCD show that there is no uniformity in the number of files changed per unit bug of future version and per unit issue of current versions. In the latest versions the values of FCQ and FCD are less as compare to the initial and middle versions. In middle versions we see high value of FCQ and FCD.

This shows that Avro product gets better code change quality over a period of time and the active contributors are producing good code change quality.

Table 8. jUDDI Dataset

Version	CCQ	CCD	FCQ	FCD
0.9rc1	0.00	0.00	0.00	0.00
0.9rc2	0.00	0.00	0.00	0.00
0.9rc3	1.00	0.00	10.00	0.00
0.9rc4	0.12	4.91	11.05	442.00
2.0rc5	0.00	0.14	0.00	11.50
2.0rc6	0.86	6.85	89.63	717.00
3.0alpha	0.00	0.49	0.00	51.21
2.0rc7	0.17	0.00	8.63	0.00
3.0beta	0.33	0.06	57.67	9.61
2.0	1.98	0.49	108.00	27.00
2.0.1	0.15	1.98	8.31	108.00
3.0rc1	0.16	0.12	37.17	27.88
3.0	0.06	0.16	13.12	37.17
3.0.1	0.14	0.11	3.43	2.82
3.0.2	0.41	0.10	5.71	1.43
3.0.3	0.98	0.25	25.00	6.25
3.0.4	0.07	0.60	0.45	4.00
3.1.0	0.49	0.11	18.67	4.23
3.1.1	0.24	0.07	8.25	2.36
3.1.2	0.00	0.00	0.00	0.00

Version	CCQ	CCD	FCQ	FCD
3.1.3	0.50	1.00	2.33	4.67
3.1.4	0.05	0.12	0.50	1.25
3.1.5	0.03	0.09	3.53	9.38
3.2	a	0.06	a	3.69

Table 9. Whirr Dataset

Version	CCQ	CCD	FCQ	FCD
0.1.0	0.32	0.11	6.08	2.09
0.2.0	0.09	0.08	1.33	1.17
0.3.0	0.13	0.04	4.93	1.50
0.4.0	0.24	0.06	6.50	1.58
0.5.0	0.42	0.09	19.71	4.06
0.6.0	0.14	0.07	2.00	1.04
0.7.0	0.43	0.08	10.56	2.02
0.7.1	0.06	0.12	2.12	4.50
0.8.0	0.80	0.07	20.33	1.79
0.8.1	0.25	0.18	3.25	2.36
0.8.2	a	0.13	a	0.52

Table 10. Pig Dataset

Version	CCQ	CCD	FCQ	FCD
0.2.0	0.07	0.01	13.27	1.48
0.3.0	0.04	0.04	4.32	4.04
0.4.0	0.49	0.05	42.33	3.91
0.5.0	0.01	0.11	2.65	29.44
0.6.0	0.03	0.03	5.39	5.54
0.7.0	0.01	0.01	1.53	1.39
0.8.0	0.17	0.02	18.15	2.62
0.8.1	0.02	0.08	2.25	9.74
0.9.0	0.1	0.01	9.68	1.02
0.9.1	0.08	0.08	3.8	3.8
0.9.2	0.02	0.07	0.95	3.48
0.10.0	0.07	0.01	2.61	0.46
0.10.1	0.04	0.19	3.29	14.55
0.11	0.07	0	1.93	0.08
0.11.1	0.01	0.12	0.35	3.76
0.12.0	a	0.02	a	1.69

Table 11. Hive Dataset

Version	CCQ	CCD	FCQ	FCD
0.3.0	0.06	0.04	4.89	3.18
0.4.0	0.58	0.03	88.70	4.29
0.4.1	0.02	0.19	1.27	13.70
0.5.0	0.02	0.01	10.70	5.70
0.6.0	0.05	0.04	4.43	3.43
0.7.0	0.96	0.02	82.60	1.54
0.7.1	0.02	0.48	1.08	33.17
0.8.0	0.73	0.02	63.88	1.57
0.8.1	0.03	0.16	0.94	5.67
0.9.0	0.01	0.01	0.66	1.12
0.10.0	0.03	0.03	3.49	2.81
0.11.0	0.01	0.01	3.13	2.46
0.12.0	0.01	0.01	3.57	5.30
0.13.0	0.26	0.01	75.95	2.46
0.13.1	0.00	0.09	0.47	15.00
0.14.0	a	0.00	a	0.13

From the results shown in table 7-11, we infer the same trends for defined metrics for jUDDI, Whirr, Pig and Hive products as Avro product.

We have also shown graphically all proposed metrics for different versions of all products. Figure 1 to 10 show the CCQ, CCD, FCQ and FCD metrics trend for the five products. We observed that all the five products follow the similar code change trend. We observed that the projects are self-regulating and self-stabilizing subject to continuing changes.

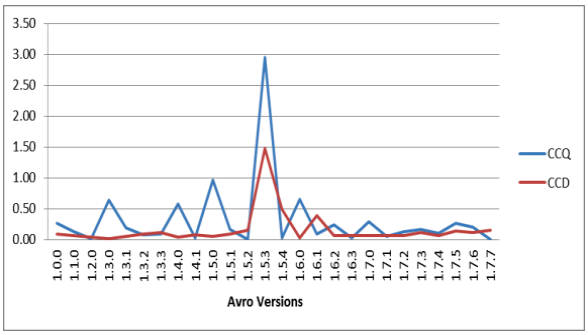

Figure 1. CCQ and CCD metrics for Avro

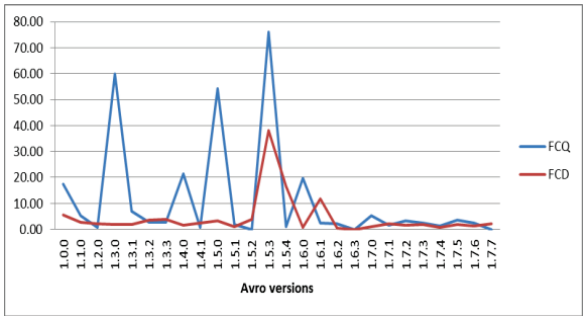

Figure 2. FCQ and FCD metrics for Avro

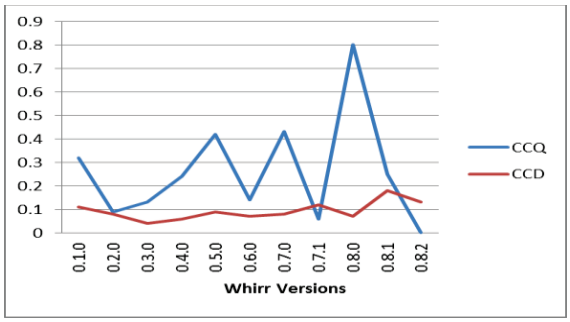

Figure 3. CCQ and CCD metrics for Whirr

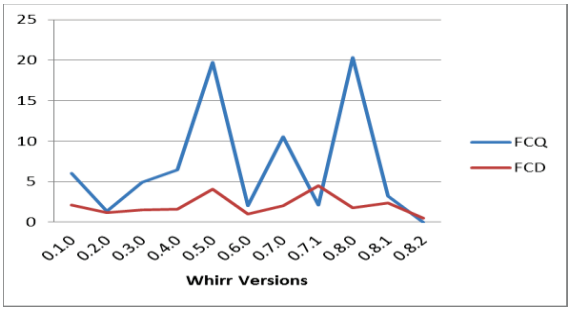

Figure 4. FCQ and FCD metrics for Whirr

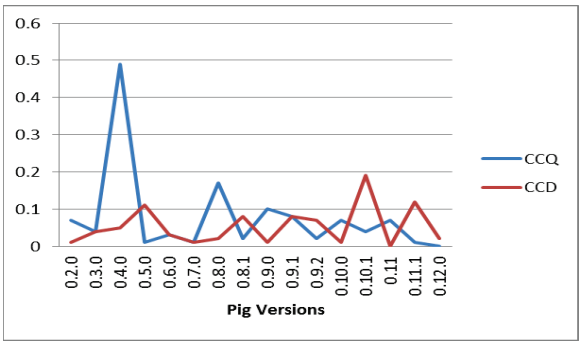

Figure 5. CCQ and CCD metrics for Pig

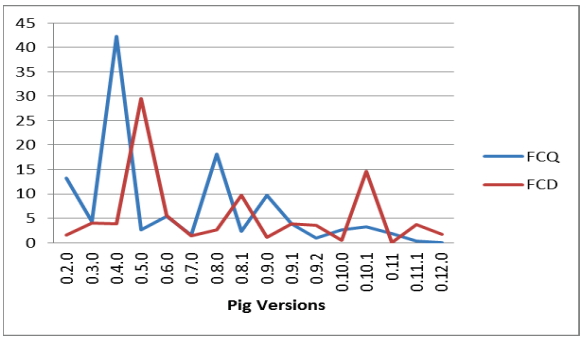

Figure 6. FCQ and FCD metrics for Pig

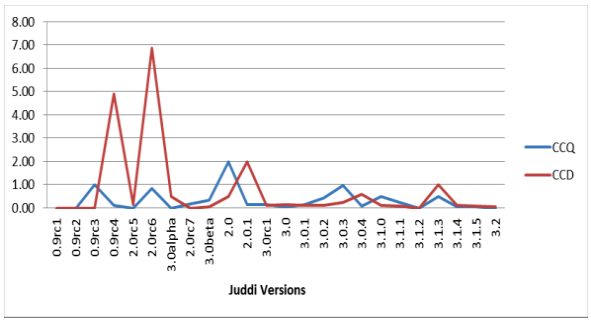

Figure 7. CCQ and CCD metrics for jUDDI

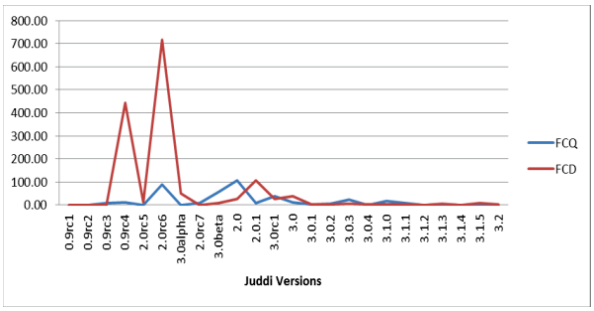

Figure 8. FCQ and FCD metrics for jUDDI

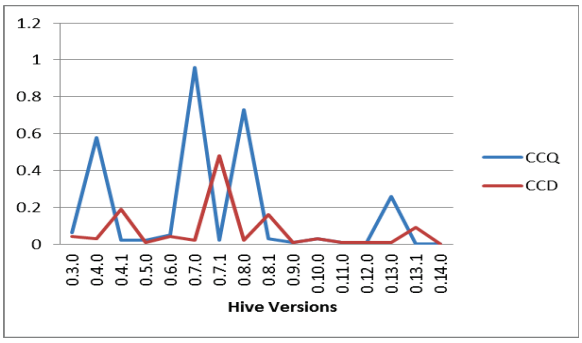

Figure 9. CCQ and CCD metrics for Hive

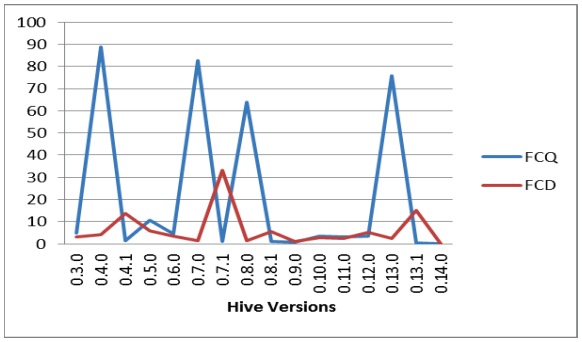

Figure 10. FCQ and FCD metrics for Hive

We have also shown graphically the distribution of different issues (bugs/improvements/new features) fixed across different versions of all products in figure 11 to 15. We observed that all products have different distribution of issues across all versions.

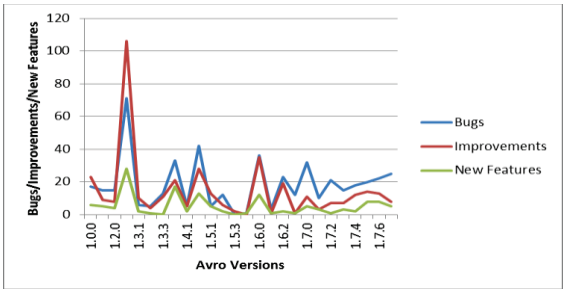

Figure 11. Issues fixed per version for Avro

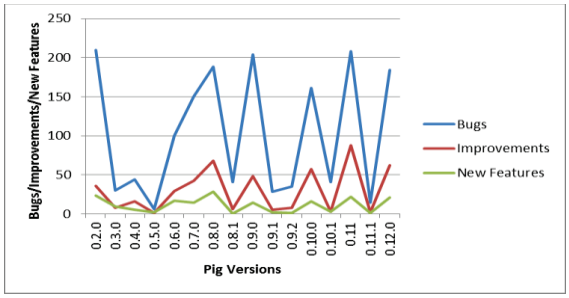

Figure 12. Issues fixed per version for Pig

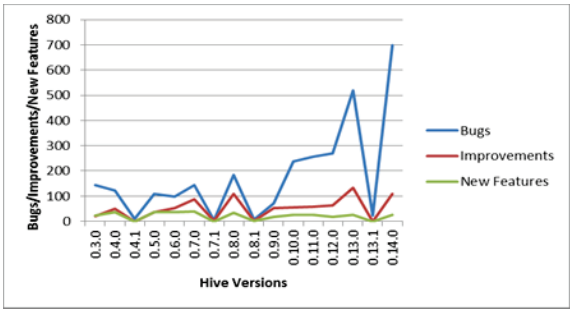

Figure 13. Issues fixed per version for Hive

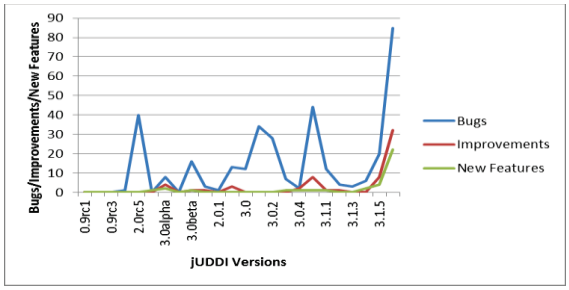

Figure 14. Issues fixed per version for jUDDI

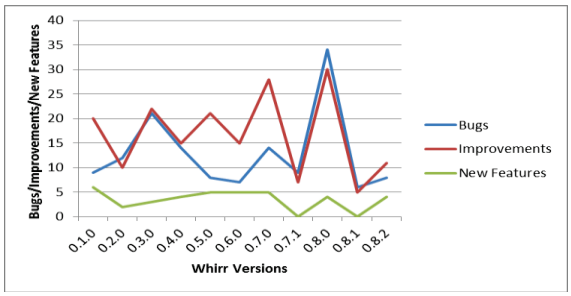

Figure 15. Issues fixed per version for Whirr

Figures 16 to 20 show cumulative issues fixed for all the products.

In all products except Whirr, we observe that number of cumulative bugs is more than improvements and new features across all versions.

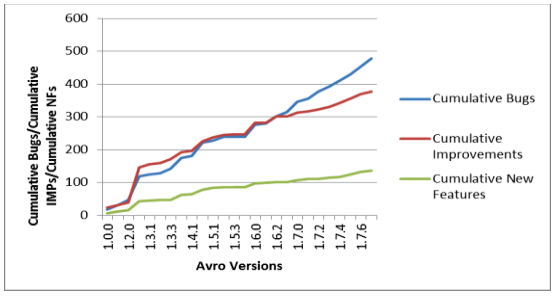

Figure 16. Cumulative Issues fixed for Avro

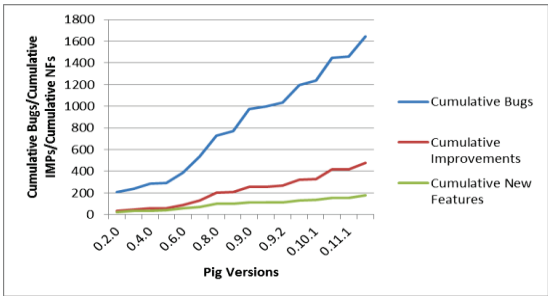

Figure 17. Cumulative Issues fixed for Pig

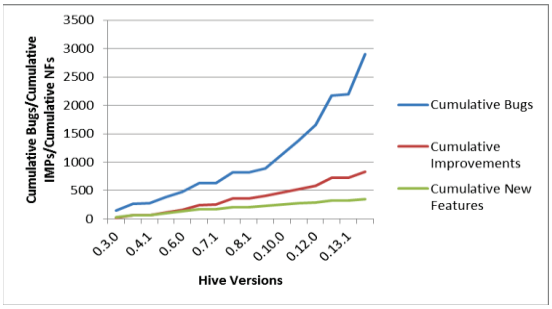

Figure 18. Cumulative Issues fixed for Hive

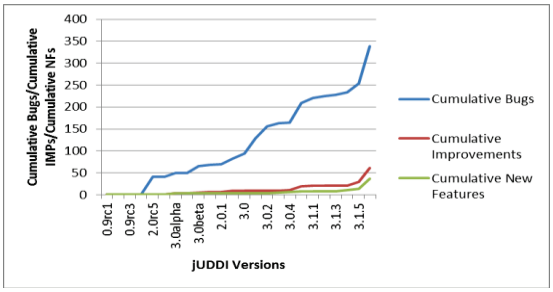

Figure 19. Cumulative Issues fixed for jUDDI

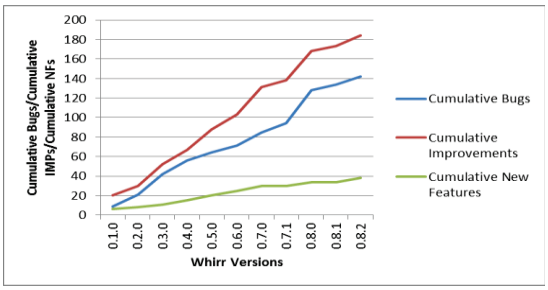

Figure 20. Cumulative Issues fixed for Whirr

7. THREATS TO VALIDITY

Following are the factors that affect the validity of our approach:

Construct Validity: We have taken combined code changes due to bugs, feature improvements and new features. We have taken both beta and stabilized versions of all products.

Internal Validity: Individual effect of bugs, feature improvements and new features on code change has not been considered.

External Validity: We have considered Apache products only which is open source. We can consider other open source and closed source software also.

8. CONCLUSION

In order to fix different issues in software, source codes of different files are changed before the next release of the product. It is important to understand the code change behavior of software across its different versions. This study will identify with the maintainability, quality and stability of the software. In this paper, we have proposed four code change quality metrics namely, CCQ, CCD, FCQ and FCD for five Apache products, Avro, Pig, Hive, jUDDI and Whirr. Using these metrics, we have quantified the quality of code and files changed during maintenance activities/evolution of the software product. For all the products, we have calculated different statistical measures, namely Minimum (Min), Maximum (Max), Average (Avg.) and the Standard Deviation (Std. Dev.) for the time gap between the two versions, the number of files changed, number of issues (bugs/improvements/new features) fixed across different versions. We observed that the average time gap between the two versions is maximum 4.83 months for jUDDI. For Hive product, we have the maximum value of average number of files changed and maximum value for average number of issues (bugs/improvements/new features) fixed in a version. All products have a different distribution of issues across all versions. In all the products except Whirr, the numbers of cumulative bugs are more than improvements and new features across all versions. We have also shown graphically proposed metrics for different versions of all the products and observed that all the five products follow the similar code change trend. The proposed metrics can be further used for other projects to understand the quality of code changes and hence the stability of the project and release readiness.

Machine learning based classifiers based on these metrics will be developed in future.

REFERENCES

[1] Hassan, A.E. 2009. Predicting faults based on complexity of code change. In *International Conference on Software Engineering*. 78-88.

[2] Chaturvedi, K.K., Kapur, P.K., Anand, S., and Singh, V.B. 2014. Predicting the complexity of code changes using entropy based measures. In *International Journal of System Assurance Engineering and Management*, Springer. 5, 155-164. doi: 10.1007/s13198-014-0226-5.

[3] Shannon, C. E. (1948), 'A Mathematical Theory of Communication', The Bell System Technical Journal, 27, 379–423,623–656.

[4] http://www.apache.org/

[5] https://github.com/

[6] Ying, A. Murphy, G. Ng, R. Chu-Carroll, M. 2004. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering* (TSE), 30,9, 574–586.

[7] Zimmermann, T., Zeller, A., Weissgerber, P. and Diehl, S. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31,6, 429 - 445.

[8] Nagappan, N. and Ball, T. 2005. Use of relative code churn measures to predict system defect density. In *Proceeding of International Conference on Software Engineering* (ICSE'05), Saint Louis MO, 284-292.

[9] Shao, D., Khurshid, S. and Perry D. E. 2009. Semantic impact and faults in source code changes: An empirical study. In *Proceeding of Software Engineering Conference*, ASWEC '09. Australian, 131-141.

[10] Pan, K., Kim, S., James, E. and Whitehead, Jr. 2009. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14, 3, 286-315.

[11] Osman, H., Lungu, M. and Nierstrasz, O. 2014. Mining frequent bug-fix code changes. In *IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering* (CSMR-WCRE), 343–347.

[12] Giger, E., Pinzger, M. and Gall, H. C. 2011. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceeding of International Workshop on Mining Software Repositories*. 83-92.

[13] Negara, S., Codoban, M., Dig, D. and Johnson, R. E. 2014. Mining Fine-Grained Code Changes to Detect Unknown Change Patterns. In *Proceeding of International Conference on Software Engineering*. In press.

[14] Kim, S., Whitehead, J. and Zhang, Y. 2008. Classifying software changes: Clean or buggy? *IEEE Transaction Software Engineering*, 34, 2, 181–196.

[15] Giger, E., Pinzger, M. and Gall, H. 2012. Can we predict types of code changes? an empirical analysis. In *MSR'12*, 217-226. IEEE CS.

[16] Tao, Y., Dang, Y., Xie, T., Zhang, D., and Kim, S. 2012. How Do Software Engineers Understand Code Changes? An Exploratory Study in Industry. In *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering* (FSE 2012). Research Triangle Park, North Carolina.

[17] Kim, M. and Notkin, D. 2009. Discovering and representing systematic code changes. In *Proceedings of International Conference on Software Engineering* ICSE'09, 309-319.

[18] Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. and Nguyen, T. N. 2009. Clone-aware configuration management. In *ASE'09*, 123-134.

[19] Singh, V.B. and Sharma, M. 2014. Prediction of the complexity of code changes based on number of open bugs, new feature and feature improvement. In Proceedings of the 25thIEEE International Symposium on Software Reliability Engineering (ISSRE), WOSD. Neples, Italy, 478-483.