Thai **J**ournal of **M**athematics : (2017) 91–105
Special Issue on Entropy in Econometrics

http://thaijmath.in.cmu.ac.th
ISSN 1686-0209

Estimating Release Time and Predicting Bugs with Shannon Entropy Measure and Their Impact on Software Quality

Talat Parveen[1] **and H. D. Arora**

Department of Applied Mathematics, Amity Institute of Applied Sciences
Amity University, Noida, Uttar Pradesh, India
e-mail : talat.tyagi@gmail.com (T. Parveen)
hdarora@amity.edu (H. D. Arora)

Abstract : As large amount of software repositories are available, the quantification of code change process is made possible and software engineering process had been paced up over a period of time. These repositories which include code change process information, bugs, and details about developers are abundantly used by researchers to fetch information which are important for improving software quality. We presume that a complex code change procedure incompatibly affects software quality. Code change process affects the quality of software and hence the software cost is affected as well. We developed a system in which data derived from the change history of each software release version is taken under consideration. This analysis shows that history change complexity metrics are prominent in predicting bugs in the software system in comparison to classical predictors of faults i.e., prior alterations, prior defects etc. Source code change data of software releases has been fetched from github repository for over a fifteen years of time, which includes 151 releases. History complexity metrics for the code change and bugs registered are used for predicting the release time and future bugs in the software release. To the data statistical multilinear regression model is utilized in predicting the software release time and estimated bugs of a software based on the history complexity metric in various releases. The performance of the model had been compared using performance R^2, $RMSE$ and MAE values.

Keywords : entropy; code change; repository; software quality; complexity met-

[1]Corresponding author.

rics.
2010 Mathematics Subject Classification : 94A17; 94A15.

1 Introduction

The modification in the code is carried out by the developer due to introduction of new features, modification in existing feature or due to fixing bugs. The changes in the code due to these reasons make source code complex and thus leading to the introduction of new faults in the system. Bug prediction in software is demanding and ever growing field of research, it identifies software modules which are more prone to have bugs before testing schedule, to reduce the testing period and to optimally allot the resources so as to reduce the total expenditure of project. The modules which are more prone to have bugs are tested exhaustively to produce high quality bug free software with low cost and for better quality assurance, distinctive and reliable bug prediction techniques are to be developed. It is also beneficial for software administrators as it supports in planning project quantitatively (Ekanayake et al. [1]).

In the closed source software, release times are fixed at the inception of the project and further versions are released at the different phases depending upon the bugs fixed and the introduction of new components according to necessities. In the open source software source codes are changed frequently with each release to fix bugs and to accomplish the objectives that concerns the users. Project administrators have a due date to be followed to release software on time with profitable financial schemes. (Gyimothy et al. [2]) estimated the effect of object oriented metrics in predicting bug, they studied the Mozilla software system which is open source software to reach to their conclusion. However with the growth of software, uncontrollable complexity grows in it which consequently influence the release schedule of the project. (Moser et al. [3]) conducted study to conclude that bug prediction is better carried out using process metrics than code metrics. (Radjenovic et al. [4]) in their review concluded that object oriented metrics are better predictor than process metrics and code metrics.

Entropy is basis of the information theory which considers probabilistic approach and focused primarily around measuring the uncertainty in the framework. In each new software release latest features are implemented to satisfy consumer demand with which project grows and hence bugs are introduced in the system with increase in entropy due to continuous code change. Unpredictability about code change can be measured utilizing entropy based metrics as done by Hassan [5]. (Hassan [5]) applied information theoretic principle to set forth the theory of complexity of code change. He applied Shannon's [6] information theory of entropy to evaluate the complexity of code change. He further assessed that History Complexity Metric (HCM) based on entropy theory can predict bugs more precisely.

We developed a system in which data derived from the change history of each software release version is taken under consideration. This analysis shows that history change complexity metrics are better predictors of fault in the software system in comparison to other well known historical predictors of faults i.e., prior modifications and prior faults. Source code change data of software releases has been fetched from github repository for over a fifteen years of time, which includes 151 releases. Entropy based history complexity metrics for the code change and bugs registered are used for predicting the release time and future bugs in the software release. To the data, after processing multi-linear regression model is applied for predicting the release time and the estimated bugs of a software based on the history complexity metric in various releases. The novel approach of predicting release time and bugs employing entropy based HCM is used. The performance of the model had been compared using performance R^2, $RMSE$ and MAE values.

2 Literature Review

Software organization have reduced the time between the new releases to meet up the requirement of the customer and inspite of releasing the full fledged release at once which would be containing the latest features and where all bugs would be fixed usually after 12 months or more, have started to release a software versions with only new features and few immediate bug fixes. With this the organizations can include 100s of new improvements and enhancements in short period of over 2 months. (Xuan et al. [7]) applied BMA (Backbone based Multilevel Algorithm), it provides optimal result by reducing the scale of subjected problem. (Beck and Andres [8]) claimed that it is beneficial for both the user and organization if release cycle of software is short, it made it possible to get the feedback faster about new improvements and fixed bugs and to implementation becomes effective in new release. (Gyimothy et al. [2]) implement object oriented code metrics as proposed by Chidamberer and Kemerer for predicting faults vulnerability, (Nagappan and Ball [9]) predicted system fault quantity utilizing code metrics. Various researchers determined that the fault potential can be predicted utilizing prior modification in the software (D'Ambros et al. [10]; Arisholm and Braind [11]; Graves et al. [12]; Khoshgoftaar et al. [13]). (D'Ambros et al. [14]) compared bug prediction methods extensively and set a benchmark in fault prediction. (Bagnall et al. [15]) proved the problem of optimal next release as N-P Hard in his work and coined the term Next Release Problem. (Garey et al. [16]) estimated that required number of next releases can never be estimated exactly through any algorithm in polynomial time. (Cheng et al. [17]) estimated that with the ever increasing user requirements, new releases of product with optimized cost is difficult to be decided. (Hassan [5] and D'Ambros [10]), has proposed bug prediction using complexity of code changes by employing linear regression technique. (Li et al. [18]) analyzed that effect on the quality of software by shifting to fast software release method. They compared the effective quality in terms of bugs fixed in fast release and usual release methods. (Hassan [5]) utilized the code change metrics and concluded that rapid code change affect the quality of software, he has proposed a method to predict bugs utilizing complexity of code changes, he also estimated that complexity metrics are better predictors than the fault potential unlike other history fault predictor such as prior changes and prior faults, he used code change data of six open source software to validate the hypothesis. The paper is organized in 5 section, Section 2 illustrates the related work. Section 3 describes the data collection method, its processing and code change metric. It illustrates the basic model of history complexity metric and its calculation method. Section 4 describes methodology and displays calculated history complexity metrics. Section 5 discusses results and paper is hence concluded.

3 Code Change Metric

Shannon [6] (1948) introduced the concept of entropy also known as measure of uncertainty in information theory attributed to his research work A mathematical theory of communication , popularly known as Shannon's entropy was described by him as:

$$H_n(P) = -\sum_{i=1}^{n}(P_i * log_2 P_i) \tag{3.1}$$

Whereas $\sum_{i=1}^{n} P_i = 1$ and $P_i \geq 0$

The probability P_i is number alteration in ith file in a particular time period by the total number of changes in all the files in considered period of time. Entropy measure as defined by Shannon is non-negative, permutationally symmetric and is additive. Also it is continuous in $0 < P_i < 1$. Entropy is maximum when all events are equally likely to occur i.e., $P_i = \frac{1}{n}, \forall i \in 1, 2, 3, ..., n$, while when each event has maximum probability of occurrence i.e., $P_i = 1$ and $\forall i \neq m$, $P_m = 0$ then the entropy is minimum. As the size of each file differ in software systems, Shannons Entropy H_n measure is normalized such that$0 \leq H_n \leq 1$ enabling the comparison of entropy measure of distributions of variant sizes, over different time period.

$$\begin{aligned}
H_n(P) &= \frac{1}{\text{(Maximum entropy for distribution)}} * H_n(P) \\
&= \frac{1}{log_2(n)} * H_n(P) \\
&= -\frac{1}{log_2(n)} * \sum_{i=1}^{n}(P_i * log_2 P_i) \\
&= -\sum_{i=1}^{n}(P_i * log_n P_i) \qquad (3.2)
\end{aligned}$$

where $P_i \geq 0$ $\forall i \in 1, 2, 3, , n$ and $\sum_{i=1}^{n} P_i = 1$

To calculate the complexity of code change in set of files for a specific period of time (year, half year, month etc.,) probability of each file is calculated and thereafter entropy is calculated using Shannon's entropy measure.

f_1	■■	■		0.4
f_2	■	■■■	■	0.2
f_3	■	■		0.2
f_4	■		■■	0.2
file/time	t_1	t_2	t_3	

Table 1: Changes in files with respect to time

P_i is the probability due to change in the i^{th} file during the defined time duration. Lets consider that in a system with 4 files total changes occurred are 13 which is divided over 3 time periods as shown in figure 1. For the time period t_1 there are total 5 changes across 4 files. The probability of change of files f_1, f_2, f_3 and f_4 would be $\frac{2}{5}(= 0.4)$, $\frac{1}{5}(= 0.2)$, $\frac{2}{5}(= 0.2)$ and $\frac{1}{5}(= 0.2)$ respectively. The Shannons entropy for time period t_1 is calculated as: =-(0.4$\log_2$ 0.4+0.2 $\log_2$ 0.2+0.2 $\log_2$ 0.2+0.2 $\log_2$ 0.2) = 1.9219576

Now using equation (2) entropy could be normalized, the value of entropy after normalization for time period t_1 is $\frac{1.9219576}{log_2 4} = 0.9609788$

3.1 History Complexity Metric

History Complexity Metric(HCM) in a system is a measure for the complexity of changes assigned to each file in the software system. History complexity metric are calculated utilizing the entropy concept in code change process, which estimates the complexity of code change in each file of the release versions. To compute HCM the History Complexity Period Factor $HCPF_i(j)$ for file during time period is calculated. For period i and entropy H_i with set of files, F_i altered with probability P_j where $j \in F_i$ here $HCPF_i$ for file j at time period i is

$$HCPF_i(j) = \begin{cases} C_{ij}H_i & j \in F_i \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where H_i represents entropy of changes during period i and C_{ij} is the contribution of entropy for period i assigned to file j, here we are considering $HCPF$ variants using varying weighting factors C_{ij}. Which are

$\boldsymbol{HCM_1}$

$\boldsymbol{HCPF_i}$ with $\boldsymbol{C_{ij}}$ =1 it assigns full complexity of each modified file i.e., equal weights for all files during $\boldsymbol{i^{th}}$ period

$\boldsymbol{HCM_2}$

$\boldsymbol{HCPF_i}$ with $\boldsymbol{C_{ij} = P_j}$ where $\boldsymbol{P_j}$ is probability of changes in file $\boldsymbol{j}$ w.r.t to modification in $\boldsymbol{i^{th}}$ period.

$\boldsymbol{HCM_3}$

$\boldsymbol{HCPF_i}$ with $\boldsymbol{C_{ij} = 1/F_i}$ where $\boldsymbol{F_i}$ is changed file in $\boldsymbol{i^{th}}$ period.

For example HCPF calculated for file $\boldsymbol{f_1}$ in time period $\boldsymbol{t_1}$ for data as shown in figure 1 , will be different for three HCM versions, for $\boldsymbol{HCM_1}$ the $\boldsymbol{HCPF}$ would be 10.9609788=0.9609788, for $\boldsymbol{HCM_2}$ the $\boldsymbol{HCPF}$ would be 0.40.9609788=0.38439152 and for HCM_3 the HCPF is 140.9609788=0.2402447. History complexity metric for a file over a period p,,q is represented as

$$HCM_{p,....,q}(j) = \sum_{i \in p,...,q} HCPF_i(j) \tag{3.4}$$

History complexity metrics imply that complexity of file over the time keep on increasing due to the modifications in the files, HCM for subsystem S over evolution period p,q is given as sum of HCM of all file:

$$HCM_{p,....,q}(S) = \sum_{i \in S} HCM_{p,....,q}(j) \tag{3.5}$$

The changes in code of open source software are random due to positioning of developers at various different locations and thus code changes and bug fixation are frequent.

3.2 Software Release Data

Bugzilla [19] (1998) is a world's leading free bug tracking system software, so different organizations uses Bugzilla, its website had listed 136 different companies which are using public Bugzilla installation and utilizing its bug tracking feature, it has begun in September, 1998 with its first release version 2.0 has so far has more than 150 releases

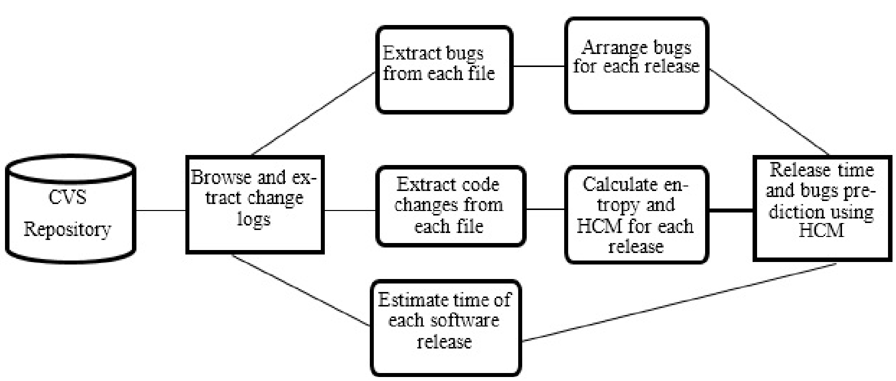

Figure 1: Data Processing Steps

Data has been prepared according to the following rules:
Step1: Release date of each software versions are noted
Step2: All logs from each release are extracted and arranged with date of change.
Step3: Changes are noted according to new feature/improvement/modification.

Step4: Total changes are recorded.
Step5: Bugs are recorded from each release.
Step6: Changes are arranged monthly and thus history complexity metric (HCM) calculated.
Step7: Time of all releases is counted in months.

The following figure 2 represents complete data in graphical form, it includes complexity of code change, time of release and bugs in each software release

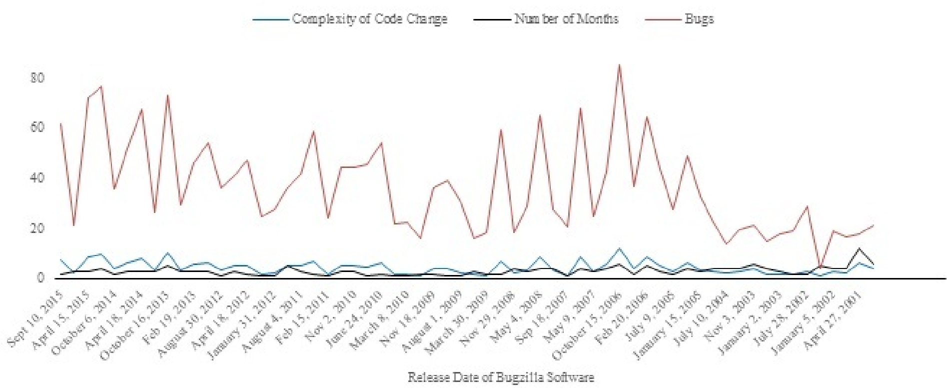

Figure 2: Software Release Data

4 Methodology

In this study bug prediction and software release time prediction model are developed utilizing history complexity metrics $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ which are calculated as explained in section 3. Regression analysis is used to establish the predicted value of a dependent variable using independent variable and value of regression coefficients which is obtained through the multiple linear regression method by using software such as $\boldsymbol{SPSS}$. The prediction model is built using the Multiple Linear Regression[20] whereas the HCM metrics are used a predictors in the model, to predict bugs and release time in months, the prediction is carried out in two stages, for Bugs($\boldsymbol{p_0}$) prediction, bugs are taken as dependent variable while HCM($\boldsymbol{q_0}$) and Time($\boldsymbol{q_1}$) are taken as independent variables. For predicting Time ($\boldsymbol{y_0}$), time is taken as independent variable whereas HCM ($\boldsymbol{x_0}$) and Bugs ($\boldsymbol{x_1}$) are kept as independent variable

$$\boldsymbol{y_0 = a_0 + a_1 x_0 + a_2 x_1} \tag{4.1}$$

$$\boldsymbol{p_0 = b_0 + b_1 q_0 + b_2 q_1} \tag{4.2}$$

In equation 4.1,$\boldsymbol{y_0}$, $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$ represents time, HCM and bugs respectively, where $\boldsymbol{a_0}$, $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are coefficients of regressions and in equation 4.2, $\boldsymbol{p_0}$, $\boldsymbol{q_0}$ and $\boldsymbol{q_1}$ represents Bugs, HCM and time respectively, where $\boldsymbol{b_0}$, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are coefficients of regressions, values of regression coefficients can be computed by applying multiple linear regression method using SPSS software, after estimating the values of regression coefficients the release time of software and the bugs can be predicted by putting the values of regression coefficients in equation 4.1 and 4.2.

Here Table 2, represents the value of $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ computed using the method explained in section 3, bugs detected, release time in months and total changes in files. These $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ values are used in equation 4.1 and 4.2 along with Bugs and Time to predict the bugs and time for future release of the software.

Total changes	$\boldsymbol{HCM_1}$	$\boldsymbol{HCM_2}$	$\boldsymbol{HCM_3}$	Bug	Time
673	7.288517	2.228849	2.235531	62	2.248006
286	2.335229	0.499643	0.405914	22	2.726052
1626	8.931459	3.403958	3.409568	73	2.726317
774	10.07652	3.054259	2.892135	77	3.930522
728	4.034633	0.901954	0.721361	36	2.467219
911	6.545516	1.698779	1.489059	52	3.303372
1542	8.360617	2.31133	2.166155	68	2.812483
302	3.296621	0.804766	0.673605	26	3.421996
2418	10.19242	2.522646	3.072282	73	5.074924
2331	3.539719	0.786362	0.39374	30	3.0589
943	5.795435	1.374203	1.168815	46	3.317826
576	6.463577	1.593775	1.502074	54	2.5627
413	3.453316	0.886149	0.821803	36	1.143094
382	5.178797	1.320193	1.209627	41	3.335681
1833	5.30215	1.411558	1.222241	48	1.895229
234	1.757591	0.309172	0.244562	25	0.663437
500	2.324726	0.366369	0.243563	28	1.080757
898	5.361164	1.27312	0.987591	37	4.918213
1254	5.252608	1.250233	1.111551	42	3.337562
827	7.040074	1.978117	1.917666	59	2.443105
342	1.629479	0.257957	0.179859	24	0.660172
734	5.377818	1.300024	1.257327	45	2.803602
891	5.464644	1.549831	1.272447	45	3.007247
1353	4.795026	1.066377	0.883691	46	1.412296
850	6.447155	1.479778	1.294096	54	2.461565
1135	1.570579	0.143268	0.072186	22	1.128679
334	1.645946	0.214263	0.136435	22	1.1306
43351	1.430351	0.583338	0.569174	16	2.400841

323	4.038138	0.728585	0.598708	37	2.265876
1506	3.925886	0.912854	0.801948	39	1.356571
509	2.552273	0.406958	0.323325	31	0.750836
210	1.939635	0.468388	0.461211	17	3.286693
312	1.426524	0.248243	0.136561	18	1.762875
1926	7.066981	1.844892	1.491319	60	2.275931
600	2.385077	2.385077	2.385077	19	3.600195
277	3.598072	0.86116	0.711607	29	3.328964
4826	8.673146	2.457049	2.164203	66	4.029041
917	3.598054	0.727378	0.549816	28	3.731827
379	1.29627	0.213824	0.124172	21	0.785967
1814	8.80902	1.605341	1.334055	68	3.663212
626	2.994401	0.624692	0.434423	25	3.179289
3448	5.583121	1.410737	1.1907	43	3.715279
4319	12.31643	2.737755	2.280997	86	6.102649
2427	4.007161	0.530275	0.132605	37	2.030083
4067	9.005605	2.434392	2.243482	65	4.910387
1595	5.234618	1.299534	1.161461	44	2.799952
1381	2.713428	0.599134	0.477675	28	1.929964
3658	6.489944	1.687377	1.39721	49	3.939824
16478	3.787442	0.687284	0.366428	33	2.763066
150	2.85968	1.140937	1.128091	23	3.511576
42387	2.105117	0.461466	0.363296	14	4.264499
378	2.690757	0.897949	0.868911	20	4.01085
10211	4.153746	1.119975	1.020463	22	6.431751
30974	1.967222	0.483385	0.318992	15	3.723833
191	2	1	1	18	3.0868
189	1.670795	0.365831	0.314746	19	2.071249
34010	2.727063	0.620164	0.39543	29	1.661735
2584	1.200153	0.288872	0.342279	4	5.047161
10891	2.784661	0.321013	0.135769	19	4.281819
137339	2.343654	0.301214	0.084696	17	4.102719
83041	6.349193	0.934531	0.411656	18	11.8928
16151	3.82533	0.64179	0.305632	22	5.75194

Table 2: Representing the computed $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ values along with total changes in files, bugs detected and time in months

5 Main Results

The result thus obtained are represented in following table, $\boldsymbol{R}$ defines the correlation between the predicted value and the observed value. It measures the strength and the direction of the linear relationship between two variables, its value lies between -1 to 1. The values of $\boldsymbol{R^2}$ obtained for predicted values of bugs and time are represented in table 3. It is observed that the value of $\boldsymbol{R^2}$ thus obtained for time using $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ are equal i.e., 0.977 for each history complexity metric predictor, thus we can say that each metric is equivalent for predicting the release time of a software. The $\boldsymbol{R^2}$ value obtained for bugs using $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ are 0.997, 0.887 and 0.705 respectively, here we can clearly estimate that the $\boldsymbol{R^2}$ value for $\boldsymbol{HCM_1}$ is best i.e., 0.997 and thus we estimate that the history complexity metric $\boldsymbol{HCM_1}$ is better estimator for predicting the bugs than $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$.

$\boldsymbol{Bug}$	$\boldsymbol{b_0}$	$\boldsymbol{b_1}$	$\boldsymbol{b_2}$	$\boldsymbol{R}$	$\boldsymbol{R^2}$	Adj. $\boldsymbol{R^2}$
$\boldsymbol{HCM_1}$	13.803	7.674	-3.671	0.998	0.997	0.997
$\boldsymbol{HCM_2}$	19.013	22.170	-2.287	0.887	0.887	0.779
$\boldsymbol{HCM_3}$	21.996	-1.579	20.184	0.839	0.705	0.695
Time	$\boldsymbol{a_0}$	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{R}$	$\boldsymbol{R^2}$	Adj. $\boldsymbol{R^2}$
$\boldsymbol{HCM_1}$	3.659	-0.274	2.127	0.988	0.977	0.976
$\boldsymbol{HCM_2}$	8.251	-0.441	9.864	0.989	0.977	0.977
$\boldsymbol{HCM_3}$	13.971	13.007	-0.641	0.988	0.977	0.976

Table 3: Values of regression coefficients and R-square values obtained w.r.t $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$

Further the performance is compared using Root Mean Square Error ($\boldsymbol{RMSE}$) and Mean Absolute Error($\boldsymbol{MAE}$). The measures $\boldsymbol{RMSE}$ and $\boldsymbol{MAE}$ are defined as

$$\boldsymbol{RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}} \tag{5.1}$$

$$\boldsymbol{MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|} \tag{5.2}$$

Where $\boldsymbol{p_i}$ represents the predicted value and $\boldsymbol{a_i}$ represents the actual value.

	$\boldsymbol{MAE_1}$	$\boldsymbol{MAE_2}$	$\boldsymbol{MAE_3}$	$\boldsymbol{RMSE_1}$	$\boldsymbol{RMSE_2}$	$\boldsymbol{RMSE_3}$
$\boldsymbol{BUG}$	0.003512	0.023732	0.027426	0.006563	0.497804	0.830838
$\boldsymbol{TIME}$	0.000237	0.0000409	0.000507	0.002109	0.002785	0.00773

Table 4: Mean absolute error and Root mean square value

Table 4 represents the result obtained for predicting software release time and bugs using $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$, where $\boldsymbol{MAE_1}$ is Mean Absolute Error value for $\boldsymbol{HCM_1}$, $\boldsymbol{MAE_2}$ is Mean Absolute Error value for $\boldsymbol{HCM_2}$ and $\boldsymbol{MAE_3}$ is Mean Absolute Error value for $\boldsymbol{HCM_3}$. In case of bugs, it is noted that for $\boldsymbol{HCM_1}$ the MAE value is minimum i.e., 0.003512 followed by $\boldsymbol{HCM_2}$(0.023732) and then $\boldsymbol{HCM_3}$(0.027426) then the RMSE value is minimum i.e., 0.006563 for $\boldsymbol{HCM_1}$ followed by $\boldsymbol{HCM_2}$(0.497804) and then $\boldsymbol{HCM_3}$(0.830838). For time, it is noted that for $\boldsymbol{HCM_2}$ the MAE value is minimum i.e., 0.0000409 followed by $\boldsymbol{HCM_1}$(0.000237) and then $\boldsymbol{HCM_3}$(0.000507) then the RMSE value is minimum i.e., 0.002109 for $\boldsymbol{HCM_1}$ followed by $\boldsymbol{HCM_2}$(0.002785) and then $\boldsymbol{HCM_3}$(0.00773). It is noticed that $\boldsymbol{HCM_1}$ is better metric to predict the bugs and time for the software releases than $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$

Table 5, represents the result obtained for predicted bugs and time through the regression, where $\boldsymbol{PB_1}$ is predicted bugs w.r.t to $\boldsymbol{HCM_1}$, $\boldsymbol{PB_2}$ is predicted bugs w.r.t to $\boldsymbol{HCM_2}$ and $\boldsymbol{PB_3}$ is predicted bugs w.r.t to $\boldsymbol{HCM_3}$ and $\boldsymbol{PT_1}$ is predicted time w.r.t $\boldsymbol{HCM_1}$, $\boldsymbol{PT_2}$ is predicted time w.r.t $\boldsymbol{HCM_2}$ and $\boldsymbol{PT_3}$ is predicted time w.r.t $\boldsymbol{HCM_3}$

$\boldsymbol{PB_1}$	$\boldsymbol{PB_2}$	$\boldsymbol{PB_3}$	$\boldsymbol{PT_1}$	$\boldsymbol{PT_2}$	$\boldsymbol{PT_3}$
61.78225	63.47141	63.70039	2.23334	2.245472	2.216601
21.56811	23.76272	25.82401	2.716369	2.70012	2.697538
72.79942	88.53255	86.71292	2.709173	2.784788	2.736269
77.30381	78.11155	74.42688	3.910522	3.931015	3.881366
35.71064	33.36804	32.66468	2.455949	2.432563	2.41568
52.16473	49.2803	46.94933	3.288178	3.275145	3.244668
68.05168	64.08029	61.45798	2.795873	2.790553	2.751618
26.49851	29.00263	30.17442	3.410323	3.399052	3.390783
74.03275	63.73377	76.27344	5.053309	5.027791	5.040897
29.71007	29.43317	25.10457	3.047423	3.027643	3.000339
46.28624	42.00821	40.43289	3.303462	3.280516	3.256291
54.22799	48.6295	48.36995	2.548559	2.526391	2.503331
36.02191	35.99076	36.74471	1.134199	1.12005	1.106836
41.43179	40.73457	41.20402	3.321991	3.309434	3.292847
47.64017	46.03817	43.72184	1.883268	1.871773	1.842986
24.59969	24.1902	25.77791	0.657082	0.63279	0.62838
27.48348	24.54347	25.12615	1.073219	1.041195	1.033167
37.08004	36.10799	34.24866	4.902265	4.885426	4.863199
41.99821	39.18357	39.22445	3.323787	3.303346	3.286074
59.1415	57.4553	56.9688	2.428468	2.425364	2.397078
23.61655	23.05478	24.47195	0.653966	0.628333	0.623912
44.91694	41.50717	43.00893	2.790377	2.769777	2.756334
44.84905	46.58799	42.99831	2.99366	2.993231	2.959806
45.46163	39.45277	37.62541	1.401534	1.371068	1.347275

54.46907	46.33087	44.33004	2.447574	2.415613	2.387755
21.45142	19.44485	21.56179	1.121932	1.089016	1.088808
22.02979	21.0188	22.8586	1.123765	1.095195	1.093245
15.72538	26.30434	29.59295	2.392602	2.404834	2.405171
36.47174	29.98186	30.50485	2.254863	2.215761	2.204782
38.91422	36.12541	36.02815	1.346864	1.324088	1.307892
30.45334	26.20565	27.26239	0.743469	0.708538	0.701295
16.45209	21.77382	26.04536	3.27673	3.268921	3.274902
18.02098	20.32371	21.86112	1.755468	1.736911	1.734275
59.95981	54.88285	48.62658	2.26148	2.245675	2.198947
18.76912	63.58073	64.40297	3.589321	3.7383	3.711399
29.17894	30.48158	31.09935	3.31707	3.303104	3.292194
66.04482	64.56648	59.52378	4.0105	4.013515	3.96605
27.7104	26.60085	27.20215	3.719411	3.694888	3.685877
20.57004	21.77142	23.13743	0.779975	0.758968	0.755018
68.43355	46.52268	43.34704	3.644992	3.569583	3.537605
25.03584	25.54402	25.7152	3.168273	3.148048	3.138101
43.18854	41.90338	40.24302	3.700639	3.687122	3.662658
86.78374	66.29197	58.77578	6.077299	6.02146	5.964655
37.09059	26.11895	21.46547	2.019411	1.96317	1.93643
65.41458	62.08211	59.75547	4.890327	4.885635	4.848712
43.81803	41.4963	41.07396	2.786894	2.769734	2.749713
27.40721	27.79804	28.53563	1.920885	1.901925	1.892783
49.4133	47.57908	44.09542	3.923868	3.912912	3.879346
32.71239	27.92272	25.02702	2.751692	2.71645	2.694812
22.77832	36.22684	39.18986	3.50028	3.529166	3.523382
14.17337	19.40961	22.54266	4.253081	4.243267	4.246542
19.6465	29.6963	33.16907	3.999098	4.012302	4.011554
22.18544	29.20604	32.49102	6.415205	6.418567	6.417415
15.07303	21.11528	22.49057	3.713271	3.707272	3.703675
17.64953	34.01707	37.23607	3.077028	3.113474	3.109681
18.79421	22.24455	24.9839	2.063166	2.049715	2.050221
28.49078	28.87414	27.29666	1.652989	1.634804	1.6172
4.291871	13.7536	20.85526	5.035752	5.035099	5.054802
19.38819	16.29576	17.95026	4.26961	4.231043	4.23083
16.61574	16.23799	17.18258	4.091241	4.061225	4.058613
19.33227	12.82119	11.72889	11.86669	11.81507	11.80719
22.11252	20.12934	19.11553	5.736648	5.704573	5.693292

Table 5: Predicted values of Bugs and Time using $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$

6 Conclusion

In this paper, History complexity metrics (HCM) as explained in section 3 of the code change are used for predicting the release time and bugs in the software release. The software entropy concept follows information theory principal which has solid mathematical establishment. There are no benchmark study utilizing software entropy based release time prediction till date however several researcher have utilized software entropy for predicting bugs in software subsystem [5]. This study also explains the procedure to calculate the entropy from a code change in various software releases. Code change process affects the quality of software and hence the software cost is affected as well. To the data statistical multilinear regression model is applied to predicting the release time and estimated bugs of a software using history complexity metric of code change for various software releases. The extensive data from the Bugzilla software releases is taken for the study, code changes in several files are recorded which are further used to compute the Shannon's entropy using which further history complexity metric measure are calculated, bugs in each release and the release time of software are noted. The history complexity metric $\boldsymbol{HCM_1}$, $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ are calculated further release time and bugs are predicted using multi linear regression model in $\boldsymbol{SPSS}$ and the performance of the model had been compared using performance $\boldsymbol{R^2}$ criteria, $\boldsymbol{RMSE}$ and $\boldsymbol{MAE}$. It is estimated that $\boldsymbol{HCM_1}$ is a better predictor of bugs than $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$ and $\boldsymbol{HCM_1}$ is a better predictor of release time than $\boldsymbol{HCM_2}$ and $\boldsymbol{HCM_3}$. In this study we have used large scale data extracted from github repository for our study in estimating bugs and release time of software, in future we are aiming for applying our data to various machine learning techniques to estimate the future bugs and release time of software. This study supports in optimizing the cost bear by testing phase of software where the information theoretic approach of entropy measure is applied to code change process.

References

[1] J. Ekanayake, J. Tappolet, H. C. Gall, A. Bernstein, Time variance and defect prediction in software projects, Empir Softw Eng 17 (4-5) (2012) 348-389 DOI:10.1007/s10664-011-9180-x.

[2] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of objectoriented metrics on open source software for fault prediction, IEEE Trans Software Eng 31 (2005) 897-910.

[3] R. Moser, W. Pedrycz, G. Succi, A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, Proceedings of the 30th international conference on software Engineering, ACM, New York, NY, USA (2008), 181-190 DOI:10.1145/1368088.1368114.

[4] D. Radjenovic, M. Herico, R. Torkar, A. Zivkovic, Software fault prediction metrics: A system-atic literature review, Information and Software Technology 55 (8) (2013) 1397-1418.

[5] A. E. Hassan, Predicting faults using the complexity of code changes, In: 31st international conference on software engineering, IEEE computer society (2009), 78-88 DOI: 10.1109/ICSE.2009.5070510.

[6] C. E. Shannon, A Mathematical Theory of Communication, Bell System Technical Journal 27 (1948) 379-423.

[7] J. Xuan, H. Jiang, Z. Ren, Z. Luo, Solving the Large Scale Next Release Problem with a Backbone Based Multilevel Algorithm, IEEE Transactions on Software Engineerin 38 (5) (2012) 1195-1212.

[8] K. Beck, Extreme Programming Explained: Embrace Change, 2nd edition, Addison Wesley, 2004.

[9] N. Nagappan, T. Ball, Use of relative code churn measures to predict system defect density, Proceedings of 27th international conference on software engineering (2005), 284-292.

[10] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empir Software Eng 17 (45) (2012) 531577.

[11] E. Arisholm, L. C. Briand, Predicting fault prone components in a java legacy system, Proceedings of the 2006 ACM-IEEE International symposium on Empirical software engineering (2006), 8-17.

[12] T. L. Graves, A. F. Karr, J. S. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Trans Softw Eng 26 (7) (2000) 653-661.

[13] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, J. P. Hudepohl, Data mining for predictors of software quality, Int J Softw Eng Knowl Eng 9 (5) (1999) 547-563.

[14] M. D'Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches, In MSR10: Proceedings of the 7th International Working Conference on Mining Software Repositories (2010), 31-41.

[15] A. Bagnall, V. Rayward Smith, I. Whittley, The Next Release Problem, Information and Software Technology 43 (14) (2001) 883-890.

[16] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, New York, NY: W. H. Freeman (1979) 109-117.

[17] B. H. C. Cheng, J. M. Atlee, Research Directions in Requirements Engineering, Proceedings of International Conference Software Engineering Workshop Future of Software Engineering (FoSE 07) (2007), 285-303 DOI:10.1109/FOSE.2007.17.

[18] P.L. Li, R. Kivett, Z. Zhan, S. Jeon, N. Nagappan, B. Murphy, A. J. Ko, Characterizing the differences between pre and post release versions of software, Proceedings of the 33rd international conference on software engineering (ICSE) (2012), 716-725 DOI: 10.1145/1985793.1985894.

[19] The Bugzilla Project (1998), http://www.Bugzilla.org.

[20] S. Weisberg, Applied Linear Regression, John Wiley and Sons, 1980.

(Received 31 August 2017)
(Accepted 30 October 2017)