WALAILAK JOURNAL
http://wjst.wu.ac.th Information Technology

Applying Information Measure for Predicting Release Time of Open Source Software

Talat PARVEEN[*] and Hari Darshan ARORA

Department of Applied Mathematics, Amity Institute of Applied Sciences, Amity University, Noida, Uttar Pradesh, India

([*]Corresponding author's e-mail: talat.tyagi@gmail.com, hdarora@amity.edu)

Received: 7 June 2016, Revised: 19 June 2017, Accepted: 28 July 2017

Abstract

Open Source Software (OSS) is updated regularly to meet the requirements posed by the customers. The source code of OSS undergoes frequent change to diffuse new features and update existing features in the system, providing a user friendly interface. The source code changes for fixing bugs and meeting user end requirements again affects the complexity of the code change and creates bugs in the software which are accountable to the next release of software. In this paper, the complexity of code changes in various Bugzilla open source software releases, from version 2.0 on 19th Sep, 1998, to 5.0.1 on 10th Sep, 2015, bugs in each software version release, and the time of release of each software version are considered, and the data used to predict the next release time. The Shannon entropy measure is used to quantify the code change process in terms of entropy for each software release. Observed code changes are utilized to quantify them into entropy units and are further used to predict the next release time. A neural network-based regression model is used to predict the next release time. The performance is compared with the R measure calculated using the multi linear regression model, and a goodness of fit curve is produced.

Keywords: Complexity of code change, entropy, prediction, software release, open source software

Introduction

Open Source Software (OSS) is popular amongst academicians and has targeted consumers in large industries. OSS is highly successful, but source code changes have to be made frequently in line with the increasingly higher demand on the customer's end.

The open source community is growing on a high scale, and various kinds of supports are provided by the OSS community, investors participate in the OSS industry by supporting ongoing projects financially, considering high income returns and new project development.

The release time of software in OSS is highly anticipated. However, it is not always possible to achieve the release of software at the defined timeline. This aspect is a matter to be explored in order to allow increased accuracy in the anticipation of OSS release time. Ngo-The [19] stated that software release planning targets the features that customers acknowledge during feedback processes, the requirements of the customers are addressed in the software release, with new features infused into a series of new product releases. Constraints, such as resources, revenue prediction, and risk needed to be managed so as to maximize profits and customer satisfaction, affect the next release time of software.

Glance [8] noted that Linux kernel releases depend on the separate testing of their submitted code, individually performed, and the final testing is done by users after the release of the software version. It has not yet been established what the criterion for the release time should be, though few projects have organized patterns for the next releases of the software.

In closed source projects, the release dates are fixed at the initial stage of the project, and consequent releases are based on the bugs fixed and the features introduced as per requirements. OSS has frequent releases, which are related to the goal achievement and are determined based on the changes in the source code and the bugs fixed. Project managers have deadlines to be followed in order to release the software on time with a profitable budget. However, as software grows in complexity, it becomes difficult to control and, thus, affects the release times, and sometimes delays project releases.

Entropy is an important basic concept of information theory, which follows a probabilistic approach and is centered mainly on measuring uncertainty in the system. Complexity of code change is measured using entropy-based metrics, as stated by Hassan [10] in his landmark research paper. New feature implementations are done in each consequent release in order to develop projects. Bugs are introduced in the project as the complexity of projects increases, due to frequent code change processes. Debuggers help in discovering bugs and resolve bug issues in released OSSs. To study the next release problem, here, the Bugzilla OSS is considered, which has had over 150 releases, starting from 1998 until September, 2015. This complexity of the code change for each release is evaluated using entropy-based metrics, and the bugs present in each software release are recorded, along with the period of each release. These are applied to neural network regression to estimate the next release time of software; also, the performance is compared with the R estimate, calculated with the multi linear regression method. This method of neural network-based regression is a novel approach, and has not yet been used to predict release time. Bugzilla releases, the complexity of code change for each release version, and bugs in each release are considered and are used as inputs for predicting the next release time.

Literature review

Various customer demands are diffused in new releases of software. It is important to address the requirements reported by users in the inclusion of new features or the updating of existing ones. OSS projects keep releasing new software versions frequently, trying to fulfil user requirements. In 2010, Hassan [10] applied information theoretic concepts in quantifying the amount of code change in terms of Shannon's entropy rule, the concept was there used in the prediction of bugs. Jain *et al*. [11] utilized the mean length, termed 'useful' codes, coined by Gsiasu and Picard, to provide generalizations of 'useful' mean length and, hence, to prove noiseless coding theorem using it. Singhal *et al*. [17] proposed and characterized the generalized entropy measure of relative information with preference, the particular case of measures was proved. Singh *et al*. [15] applied Simple Linear Regression using complexity of code change and detected bugs in order to anticipate future bugs. Xuan *et al*. [23] addressed the next release problem using Backbone-based Multilevel Algorithm (BMA). BMA can be applied to get better result in large scale NRP, it can reduce the scale of the problem and build an optimal solution. D'Ambros *et al.* [5] compared bug prediction methods extensively, and set a benchmark in fault prediction. Singh *et al.* [16] applied support vector regression in predicting the bugs, using an entropy measure in the system for a set duration of time. Garzarelli [7] explained the organizational structure of OSS, and that it works without any ownership or hierarchy.

Bagnall *et al*. [1] modeled the problem of optimal next release as NP-Hard in his work, and coined the term 'Next Release Problem'. Greer *et al.* [9] utilized a genetic algorithmic approach in optimizing the release time of software versions. Garey *et al.* [6] suggested that the required number of next releases can never be estimated exactly through any algorithm in a polynomial period of time. Cheng *et al.* [3] suggested that, with ever increasing user requirements, it is difficult to decide on optimized costs for new releases of products. Ngo-The *et al.* [19] combined integer programming with 2-phased optimization to release search space and used genetic programming to minimize search space. Baker *et al.* [2] addressed the next release problem, utilizing the greedy and simulating annealing algorithm. Jiang *et al.* [12] designed the Hybrid Ant Colony Optimization algorithm (HACO) to solve the next release problem, and concluded that HACO gives better results than the existing GRASP and simulated annealing algorithm. Kapur *et al.* [13] proposed a method for the release time problem utilizing reliability, bugs fixed, and cost. Chaturvedi *et al*. [4] utilized complexity of code change and bugs in estimating the next release time of software.

The paper is organized into 7 sections, Section 2 illustrates the related work. Section 3 describes the code change process used in this paper to quantify changes in code in terms of entropy, and illustrates the basic model for entropy calculation. Section 4 explains data collection and the method of processing the data. Section 5 describes the neural network-based regression model and the multi linear regression model used to predict the next release time of the Bugzilla OSS. Section 6 describes the results and discusses them. The paper is concluded with Section 7.

Code change process

The Code change procedure represents the patterns of modifications made to source code. The modification in the code is carried out by the developer, due to the introduction of new features, to the modification of existing features, or to fixing bugs. The changes in the code for the above-stated reasons make source codes complex and, thus, leads to the introduction of new faults in the system. There may be a delay in the next release of software if the software developer fails to understand the code change process appropriately. It may also affect the quality of the software system. The code change procedure is recorded in big CVS repositories to manage it correctly. There are 3 types of modification process, following Hassan [10]. Fundamental code change technique evaluates examples of changes, as opposed to measuring the quantity of changes, or measuring the impact of changes to the code structure. The progressions are recorded, taking into account the quantity of times the document is adjusted. These progressions are measured at the document level, rather than at the code level.

Entropy is ascertained in view of the quantity of changes in a record for particular periods. The period can be taken over a day, week, month, year, or so on, in light of the aggregate length of time of the venture and, in addition, the quantity of changes happens in the framework. At the Broadened Code Change (BCC) level, instead of utilizing a settled length period, the fundamental code change is augmented in light of a variable length period. This time period can be isolated in 3 ways, i.e., time-based periods, change breaking point-based periods, and burst-based periods. In time-based periods, the downright length of the task is separated into an equivalent length span. These allotments can be of any length. In change breaking point-based periods, the periods are decided in light of the equivalent number of alterations. The progressions do not take after a particular example; instead, it, by and large, takes after the burst-based examples. The burst-based period depends on examples of the progressions happening in the undertaking, rather than in the period-based or change breaking point-based periods. The files which are changed during the high complexity leads to the introduction of new faults in the system.

Complexity of code change

Shannon [18], in 1948, introduced the concept of entropy, also known as the "measure of uncertainty", to information theory, attributed to his research work "A mathematical theory of communication ", popularly known as Shannon's Entropy, it was described as;

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

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

The probability P_i is a number alteration in the i[th] file in a particular time period by the total number of changes in all the files in a considered period of time. The entropy measure as defined by Shannon is non-negative, permutationally symmetric, and additive. Also, it is continuous in $0 < P_i < 1$. Entropy is at maximum when all events are equally likely to occur, i.e., $P_{i=}\frac{1}{n}, \forall\, i \in 1,2,3, \ldots \ldots n$, when each event has a maximum probability of occurrence, i.e., $P_i = 1$ and $\forall\, i \neq m, P_m = 0$, then the entropy is at minimum.

As the size of each file differs in software systems. Shannon's Entropy H_n measure is normalized such that $0 \leq H_n \leq 1$ enables the comparison of entropy measures of distributions of variant sizes, over different time periods.

$$H_n(P) = \frac{1}{Mazimum\ entropy\ for\ distribution} * H_n(P) \quad (2)$$

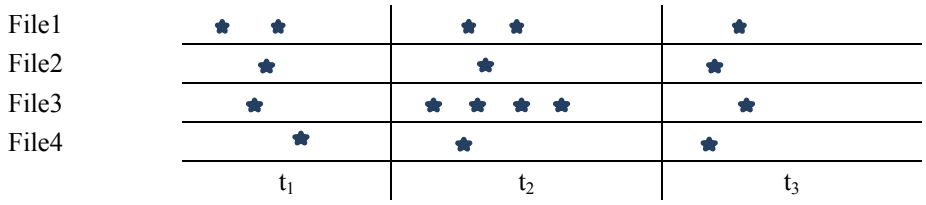
$$= \frac{1}{log_2 n} * H_n(P) = -\frac{1}{log_2 n} * \sum_{i=1}^{n}(P_i * log_2 P_i)$$

$$= -\sum_{i=1}^{n}(P_i * log_n P_i)$$

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

To study code change process information, a theoretic approach to evaluate complexity/uncertainty could be employed. To calculate the complexity of code change in a set of files for a specific period of time (year, half year, month, etc.,), the probability of each file is calculated and, thereafter, entropy is calculated using Shannon's entropy measure.

Table 1 Changes in File1, File2, File3, and File4, with respect to time period t_1, t_2 and t_3.



Consider a system with 4 files, in which changes have occurred over a period of time, these changes are noted in **Table 1** with star marks. Let there be a total of 17 changes in all 4 files. For each file, the number of changes in a file is divided by the total changes in all the files over the particular time period. For time period t1, file1 has 2 changes, file2 has 1 change, file3 has 1 change, and file4 has one change. So, the probability of file1 for t1 is 2/5 = 0.4, of file2 for t1, 1/5 = 0.2, of file3 for t1, 1/5 = 0.2 and of file4 for t1, 1/5 = 0.2. Similarly, the probabilities for each time period could be calculated and, thus, entropy/complexity of code change for each time period can be calculated. When there are changes in all files, the entropy would be maximum, while it would be minimum for most changes occurring in a single file.

Data collection and methodology

Bugzilla [20] is the world's leading free bug-tracking system software; it tracks bugs, communicates with teammates, and manages quality assurance. Bugzilla, unlike its counterparts, is free, and allows developers to follow all bug issues in their projects easily. It is under constant development, and has a dedicated team. It offers many features to the users, such as automatic duplicate bug detection, reports and charts, bug lists in multiple formats, the option to file/modify bugs by email, etc.; also, it offers excellent features to administrators, such as the ability to impersonate users, excellent security, multiple authentication methods, and compatibility with many operating systems. It is mainly used by deployment managers in chip design problem tracking, software bug tracking, and system administration. Many different organizations and projects use Bugzilla, the Bugzilla website lists 136 different companies which use public Bugzilla installation and utilize its bug tracking feature, such as Mozilla, GNOME,

W3C, LibreOffice, GCC, Linux kernel, Open Office, Scilab, Eclipse, Red Hat, KDE, Novell Inc., North Carolina State University LUG, and Apache Project. The Bugzilla project began in September, 1998, under the name Netscape, with its first release version 2.0, and up to now has had 150 releases, the latest release is 5.0.1, which was released on 10th September 2015. The date is collected from Bugzilla website for each software version release, the number of files in each software version release is observed, and changes in the code in each file are recorded; hence, the complexity of code change, as stated by Hassan [10], is calculated for each release. The collected data is processed on a monthly basis, and the changes in each release are arranged month wise; thus, entropy is calculated for each release on a monthly basis. Additionally, the bugs and total changes in each release are recorded. A snapshot from the Bugzilla repository for the Bugzilla-4.4.1 version release, depicting the bugs and other changes, along with the dates of the changes of code, is shown in **Figure 1**.

2013-10-17	Byron Jones	Bug 927736: "invalid token" error if someone else chang... bugzilla-4.4.1 release-4.4.1	commit | commitdiff | tree | snapshot
2013-10-17	Byron Jones	Bug 927570: mid-air conflict fails to check all changed...	commit | commitdiff | tree | snapshot
2013-10-16	Dave Lawrence	Bump version to 4.4.1	commit | commitdiff | tree | snapshot
2013-10-16	Frédéric Buclin	Bug 924932: (CVE-2013-1743) [SECURITY] Field values...	commit | commitdiff | tree | snapshot
2013-10-16	Frédéric Buclin	Bug 924802: (CVE-2013-1742) [SECURITY] (XSS) "id" and...	commit | commitdiff | tree | snapshot
2013-10-16	Frédéric Buclin	Bug 913904: (CVE-2013-1734) [SECURITY] CSRF when updati...	commit | commitdiff | tree | snapshot
2013-10-16	Frédéric Buclin	Bug 911593: (CVE-2013-1733) [SECURITY] CSRF in process_...	commit | commitdiff | tree | snapshot
2013-10-16	Dave Lawrence	Bug 907438 - In MySQL, login cookie checking is not...	commit | commitdiff | tree | snapshot
2013-10-16	Dave Lawrence	Bug 906745 - In MySQL, tokens are not case-sensitive...	commit | commitdiff | tree | snapshot
2013-10-16	Frédéric Buclin	Bug 912641: Release notes for Bugzilla 4.4.1	commit | commitdiff | tree | snapshot
2013-10-15	Byron Jones	Bug 917370: large dependency trees are very slow to...	commit | commitdiff | tree | snapshot
2013-10-11	Frédéric Buclin	Fixes on checkin for bug 769134	commit | commitdiff | tree | snapshot
2013-09-30	Dave Lawrence	Bug 864625 - Setting a non-privileged user as a request...	commit | commitdiff | tree | snapshot
2013-09-28	Frédéric Buclin	Bug 891311: Text in the "My Requests" page is misleadin...	commit | commitdiff | tree | snapshot
2013-09-27	Frédéric Buclin	Bug 851267: Bugzilla times out when a user has several...	commit | commitdiff | tree | snapshot
2013-09-26	Frédéric Buclin	Bug 920787: The "Flags:" label in bug reports is badly...	commit | commitdiff | tree | snapshot
2013-09-26	Simon Green	Bug 893589 - 004template.t fails when templates in...	commit | commitdiff | tree | snapshot
2013-09-26	Simon Green	Bug 769134 - Bugzilla unintentionally removes groups...	commit | commitdiff | tree | snapshot
2013-09-25	Dirk Steinmetz	Bug 455301: Don't show password box on userprefs.cgi...	commit | commitdiff | tree | snapshot
2013-09-24	Frédéric Buclin	Temporary fix for bug 916882: whitelist product and...	commit | commitdiff | tree | snapshot
2013-09-23	Jiří Netolický	Bug 919475: [Oracle] Crash when non-mandatory free...	commit | commitdiff | tree | snapshot
2013-09-18	Byron Jones	fix typo in comment	commit | commitdiff | tree | snapshot
2013-09-18	Byron Jones	Bug 877545: quicksearch shouldn't treat apostrophes...	commit | commitdiff | tree | snapshot
2013-09-16	Frédéric Buclin	Bug 785565: Search by change history between two dates...	commit | commitdiff | tree | snapshot

Figure 1 Snapshot of reported bugs for Bugzilla-4.4.1 version.

Data has been prepared according to the following rules:

- Bugzilla software versions release dates are noted from the website.
- All the reported logs are noted from each version, along with the date of change.
- Corresponding to each version, changes in the code are noted from the website for each reported bug/modification/new feature addition.
- The total number of changes in each version are recorded.
- Bugs are noted separately for each software version
- In each version, the changes are arranged month-wise and, hence, complexity of code change is also calculated monthly for each release.

The time for each release is also calculated in months.

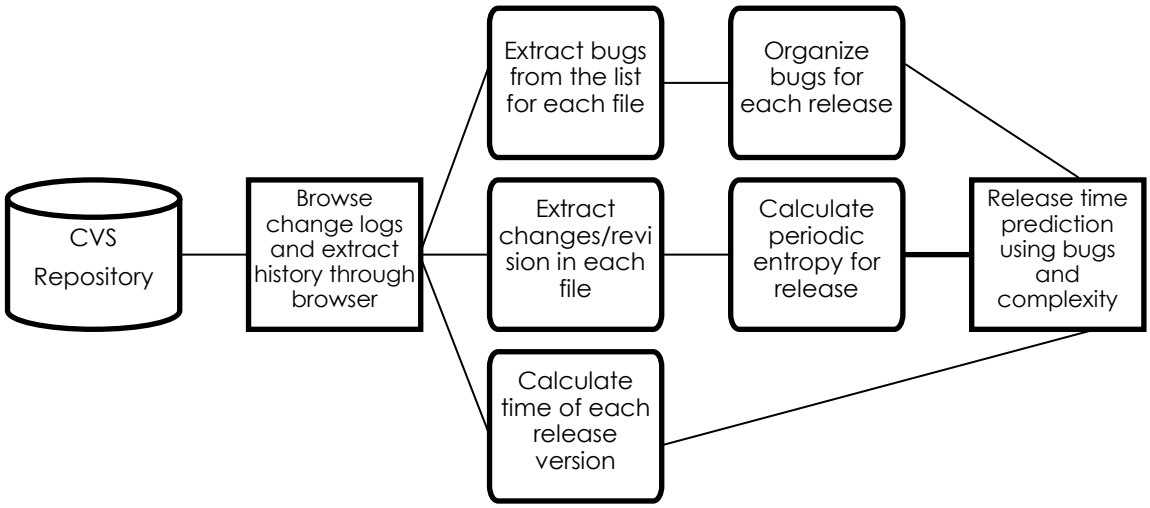


Figure 2 Process of extracting data from resources.

We have calculated the code change complexity, following the method described in section 3.1, and bugs in each release of the Bugzilla OSS and, thus, predicted the next release time of software. Neural network regression is applied to predict the next release time. **Table 2** consists of the entirety of the software versions, with their release dates and complexity of code changes in each release, and contains the number of files from which total changes in the version, in terms of code, is recorded. The code changes in each file is taken into consideration to calculate the Complexity of Code Change (COCC), using Shannon's entropy measure, to quantify the code change statistics into entropy units, and **Figure 3** illustrates the bug change patterns in all the releases of the Bugzilla OSS.

Table 2 Complexity of code change of various software releases, along with release dates and file changes.

SV	DOR	COCC	TNOF	NM	TC	SV	DOR	COCC	TNOF	NM	TC
5.0.1	Sep 10, 2015	7.288517	35	2	673	**3.2.4**	July 8, 2009	1.939635	16	3	210
5.0	July 7, 2015	2.335229	21	3	286	**3.2.3**	Mar 30, 2009	1.426524	21	2	312
4.4.9	Apr 15, 2015	8.931459	17	3	1626	**3.2.2**	Feb 3, 2009	7.066981	5	2	1926
4.4.8	Jan 27, 2015	10.07652	5	4	774	**3.2**	Nov 29, 2008	2.385077	20	4	600
4.4.6	Oct 6, 2014	4.034633	20	2	728	**3.0.5**	Aug 12, 2008	3.598072	28	3	277
4.4.5	July 24, 2014	6.545516	26	3	911	**3.0.4**	May 4, 2008	8.673146	74	4	4826
4.4.4	Apr 18, 2014	8.360617	4	3	1542	**3.0.3**	Jan 8, 2008	3.598054	43	4	917
4.4.2	Jan 27, 2014	3.296621	24	3	302	**3.0.2**	Sep 18, 2007	1.29627	22	1	379
4.4.1	Oct 16, 2013	10.19242	67	5	2418	**3.0.1**	Aug 23, 2007	8.80902	88	4	1814
4.4	May 22, 2013	3.539719	36	3	2331	**3.0**	May 9, 2007	2.994401	46	3	626

SV	DOR	COCC	TNOF	NM	TC	SV	DOR	COCC	TNOF	NM	TC
4.2.5	Feb 19, 2013	5.795435	23	3	943	**2.22.2**	Feb 2, 2007	5.583121	48	4	3448
4.2.4	Nov 13, 2012	6.463577	37	3	576	**2.22.1**	Oct 15, 2006	12.31643	129	6	4319
4.2.3	Aug 30, 2012	3.453316	24	1	413	**2.22**	Apr 22, 2006	4.007161	88	2	2427
4.2.2	July 26, 2012	5.178797	27	3	382	**2.20.1**	Feb 20, 2006	9.005605	125	5	4067
4.2.1	Apr 18, 2012	5.30215	50	2	1833	**2.20**	Sep 30, 2005	5.234618	60	3	1595
4.2	Feb 22, 2012	1.757591	10	1	234	**2.18.3**	July 9, 2005	2.713428	3	2	1381
4.0.4	Jan 31, 2012	2.324726	16	1	500	**2.18.1**	May 11, 2005	6.489944	60	4	3658
4.0.3	Dec 28, 2011	5.361164	35	5	898	**2.18**	Jan 15, 2005	3.787442	123	3	16478
4.0.2	Aug 4, 2011	5.252608	51	3	1254	**2.16.7**	Oct 24, 2004	2.85968	10	4	150
4.0.1	Apr 27, 2011	7.040074	29	2	827	**2.16.6**	July 10, 2004	2.105117	37	4	42387
4.0	Feb 15, 2011	1.629479	24	1	342	**2.16.5**	Mar 3, 2004	2.690757	20	4	378
3.6.4	Jan 24, 2011	5.377818	26	3	734	**2.16.4**	Nov 3, 2003	4.153746	34	6	10211
3.6.3	Nov 2, 2010	5.464644	40	3	891	**2.16.3**	Apr 25, 2003	1.967222	39	4	30974
3.6.2	Aug 5, 2010	4.795026	48	1	1353	**2.16.2**	Jan 2, 2003	2.0	4	3	191
3.6.1	Jun 24, 2010	6.447155	32	2	850	**2.16.1**	Sep 30, 2002	1.670795	12	2	189
3.6	Apr 13, 2010	1.570579	56	1	1135	**2.16**	July 28, 2002	2.72706	52	2	34010
3.4.6	Mar 8, 2010	1.645946	27	1	334	**2.14.2**	Jun 7, 2002	1.200153	17	5	2584
3.4.5	Jan 31, 2010	1.430351	24	2	43351	**2.14.1**	Jan 5, 2002	2.784661	175	4	10891
3.4.4	Nov 18, 2009	4.038138	8	2	323	**2.14**	Aug 29, 2001	2.343654	134	4	137339
3.4.2	Sep 11, 2009	3.925886	36	1	1506	**2.12**	Apr 27, 2001	6.349193	298	12	83041
3.4.1	Aug 1, 2009	2.552273	13	1	509	**2.10**	May 9, 2000	3.82533	206	6	16151

SV~Software Versions, DOR~Date of release, COCC~Complexity of code change, TNOF~Total Number of Files, NOM~Number of Months, TC~Total Changes.

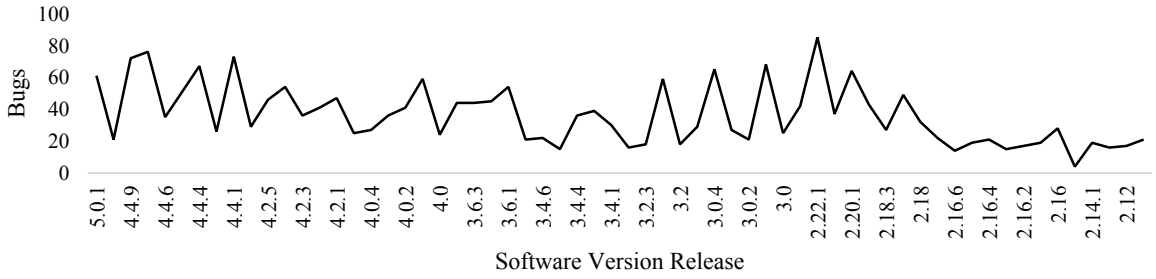


Figure 3 Bugs in Bugzilla software releases.

Neural network regression and multiple linear regression

A neural network fitting tool is applied to fit data, using the neural network toolbox in MATLAB software. Regression analysis is carried out using complexity of code changes and bugs as predictors for

the release time estimation of new releases. Fitting tool 'nnft', or neural network fitting tool, is used to perform regression on the data, consisting of complexity of code changes, bugs fixed, and time of each release in Bugzilla OSS. Complexity of code is calculated for each software release using entropy based metrics, as discussed in section 3.1. Data input is selected using the 'nnft', it is trained, and the performance is evaluated using the neural network-based regression method. The 'nnft' creates a 2 layer feed forward network with sigmoid hidden neurons and linear output neurons. The neural network is trained using the Levenberg-Marquardt back-propagation algorithm, containing 15 hidden neurons. The plot of regression is generated by MATLAB software, as shown in **Figure 5**.

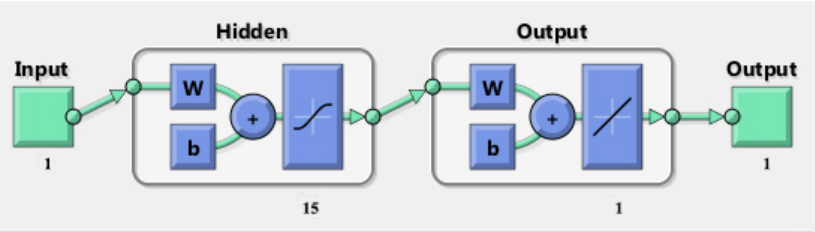


Figure 4 Neural network structure.

Multiple linear regression method is used to predict release time, considering the complexity of code change (X_1) and the bugs fixed (X_0) as independent variables, while release time (Y_0) in months as a dependent variable;

$$Y_0 = a + bX_0 + cX_1 \quad (3)$$

where *a, b* and *c* are regression coefficients, and values of *a, b* and *c* can be estimated using the multilinear regression analysis method. After estimating regression coefficients, the next release times of software can be predicted.

Results and discussion

R defines the correlation between the observed and the predicted values. It measures the strength and the direction of a linear relationship between 2 variables, and it lies between –1 to 1. The values of R, calculated through neural network-based regression and multi linear regression analysis, are depicted in **Table 3**, it is observed that the values of R, as estimated through neural network-based regression, is 0.99, while it is 0.98 when estimated through multi-linear regression analysis using SPSS. Hence, it is estimated that neural network-based regression provides better results than the multi-linear regression analysis method.

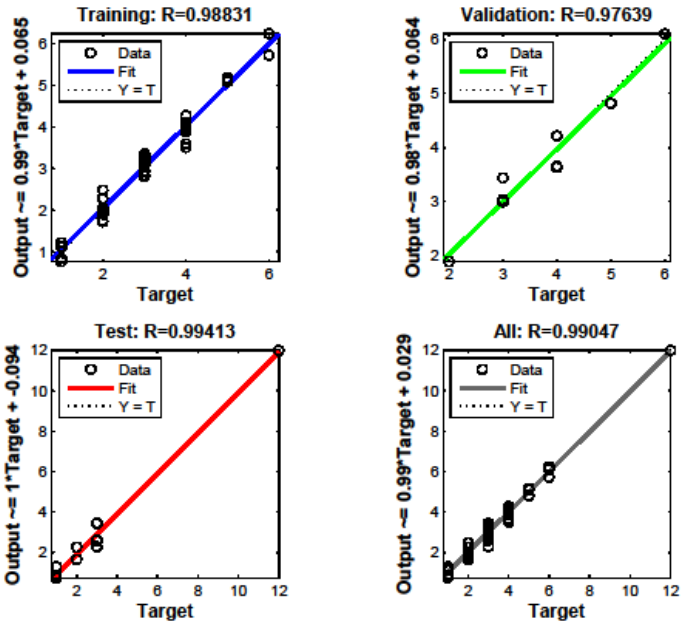


Figure 5 Neural network regression results for release time prediction model.

Table 3 depicts the values of R for neural network-based regression and multi linear regression.

Table 3 Parameters of neural network.

Method	R
Neural network regression	0.99
Multi linear regression	0.98

A goodness of fit curve has been plotted between the observed and predicted values, **Figure 5** depicts the fitting between the next release observed value and the next release predicted value.

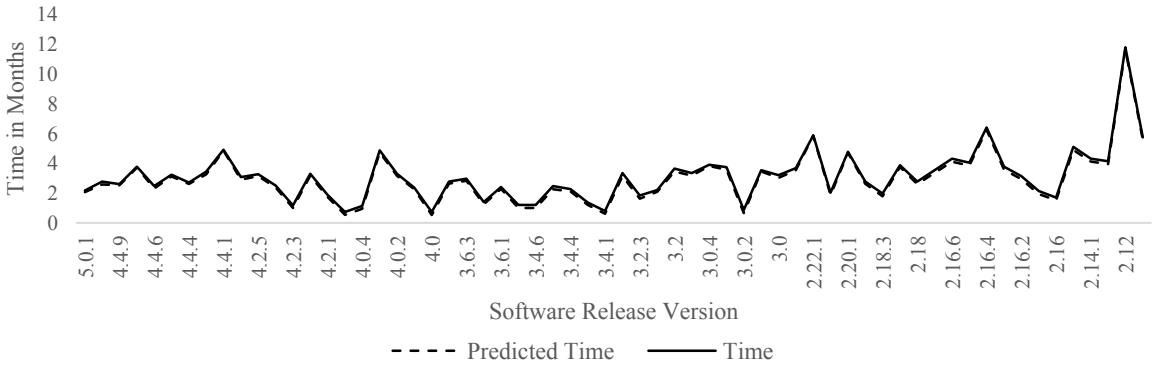


Figure 6 Goodness of fit curve for different software release versions.

It is observed that the predicted and observed values are almost similar for each software release where, in data preparation, we have merged the software releases which were released on the same day. The total releases of Bugzilla software number 150 but, as many of them were merged as they had been released on the same day, there were few initial releases from our study, as there were no bugs reported in them, our study was reduced to a total of 62 observations. The software released which were merged together due to having the same release dates are depicted in **Table 4**.

Table 4 Software releases list which have been merged together.

Main SR	**Merged Software Releases**					***Main SR***	**Merged Software Releases**			
5.0.1	*4.4.10*	*4.2.15*				**3.6.1**	3.4.7			
4.4.9	4.2.14	4.0.18				**3.4.5**	3.2.6	3.0.11		
4.4.8	4.2.13	4.0.17	4.4.7	4.2.12		**3.4.4**	3.4.3	3.0.10		
4.4.6	4.2.11	4.0.15				**3.4.2**	3.2.5	3.0.9		
4.4.5	4.2.10	4.0.14				**3.4.1**	3.4	3.2.6	3.0.11	
4.4.4	4.2.9	4.0.13	4.4.3	4.2.8	4.0.12	**3.2.2**	3.2.1	3.0.8	3.0.7	2.22.7
4.4.1	4.2.7	4.0.11				**3.2**	3.0.6	2.22.6	2.20.7	
4.4	4.2.6					**3.0.5**	2.22.5			
4.2.5	4.0.10					**3.0.4**	2.22.4	2.20.6		
4.2.4	4.0.9					**3.0.1**	2.22.3	2.20.5		
4.2.3	4.0.8					**2.22.2**	2.20.4			
4.2.2	4.0.7					**2.22.1**	2.20.3	2.18.6		
4.2.1	4.0.6					**2.22**	2.20.2			
4.2	4.0.5					**2.20.1**	2.18.5	2.16.11		
4.0.4	3.6.8					**2.20**	2.18.4			
4.0.3	3.6.7					**2.18.3**	2.18.2			
4.0.2	3.6.6					**2.18.1**	2.16.10	2.16.9		
4.0.1	3.6.5					**2.18**	2.16.8			
3.6.4	3.4.10					**2.16.2**	2.14.5			
3.6.3	3.4.9					**2.16.1**	2.14.4			
3.6.2	3.4.8					**2.16**	2.14.3			

Conclusions

In our paper, we have developed an approach to determine the predicted time of the next release of the open source software Bugzilla, using the neural network-based regression method and multi linear regression. The data was collected from the Bugzilla website, www.Bugzilla.org, for each software release. Code changes in each release were noted and, hence, complexity of code changes were calculated for each release using Shannon's entropy measure, bugs reported in each release were recorded, and time of each release was noted. These statistics were used to predict the next release times of Bugzilla software versions. Neural network-based regression and multi linear regression were carried out, and the performance of both models compared using R statistics, it was found that neural network-based regression results were better than those that were produced using the multi linear regression model at predicting the release times of software. This study can further be extended by predicting or estimating the next release times of software versions using other prediction models and other regression methods. Additionally, this method can be applied to predict the next release times of other OSS projects.

References

[1] A Bagnall, V Rayward-Smith and I Whittley. The next release problem. *Inform. Software Tech*. 2001; **43**, 883-90.
[2] P Baker, M Harman, K Steinhofel and A Skaliotis. Search based approaches to component selection and prioritization for the next release problem. *In*: Proceedings of the 22nd IEEE International Conference Software Maintenance. Philadelphia, USA, 2006, p. 176-85.
[3] BHC Cheng and JM Atlee. Research directions in requirements engineering. *In*: Proceedings of the International Conference Software Engineering Workshop Future of Software Engineering. Washington DC, USA, 2007, p. 285-303.
[4] KK Chaturvedi, P Bedi, S Mishra and VB Singh. An empirical validation of the complexity of code changes and bugs in predicting the release time of open source software. *In*: Proceedings of the IEEE 16th International Conference on Computational Science and Engineering. Sydney, Australia, 2013, p. 1201-6.
[5] MD Ambros, M Lanza and R Robbes. An extensive comparison of bug prediction approaches. *In*: Proceedings of the 7th International Working Conference on Mining Software Repositories. South Africa, 2010, p. 31-41.
[6] MR Garey and DS Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979, p. 109-17.
[7] G Garzarelli. *Open Source Software and the Economics of Organization. In*: J Bimer and P Garrouste (eds.). Markets, Information and Communication, Routledge, New York, 2004, p. 47-62.
[8] DG Glance. Release criteria for the Linux kernel. *First Monday* 2004; **9**, 1056.
[9] D Greer and G Ruhe. Software release planning: An evolutionary and iterative approach. *Inform. Software Tech*. 2004; **46**, 243-53.
[10] AE Hassan. Predicting Faults based on complexity of code change. *In*: Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009, p. 78-88.
[11] P Jain and RK Tuteja. On coding theorem connected with 'useful' entropy of order-β. *Int. J. Math. Math. Sci.* 1989; **12**, 193-8.
[12] H Jiang, J Zhang, J Xuan, Z Ren and Y Hu. A hybrid ACO algorithm for the next release problem. *In*: Proceedings of the 2nd International Conference on Software Engineering and Data Mining. China, 2010, p. 166-71.
[13] PK Kapur, VB Singh, OP Singh and JNP Singh. Software release time based on multi attribute utility functions. *Int. J. Reliab. Qual. Saf. Eng*. 2013; **20**, 1350012.
[14] MATLAB version 8.3. *Natick. IEEE Software*. The Mathworks, Massachusetts, 2014.
[15] VB Singh and KK Chaturvedi. *Improving the Quality of Software by Quantifying the Code Change Metric and Predicting the Bugs. In*: B Murgante, S Misra, M Carlini, CM Torre, HQ Nguyen, D Taniar, BO Apduhan and O Gervasi (eds.). Computational Science and Its Applications, 2013, p. 408-26.
[16] VB Singh and KK Chaturvedi. Entropy based bug prediction using support vector regression. *In*: Proceedings of the 12th International Conference on Intelligent Systems Design and Applications. Kochi, India, 2012, p. 746-51.
[17] LC Singhal, RK Tuteja and P Jain. On measures of relative information with preference. *Comm. Stat. Theor. Meth.* 1988; **17**, 1449-64.
[18] CE Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.* 1948; **27**, 379-423.
[19] A Ngo and G Ruhe. Optimized resource allocation for software release planning. *IEEE Trans. Software Eng.* 2009; **35**, 109-23
[20] The Bugzilla Project, Available at: http://www.Bugzilla.org, accessed January 2016.
[21] G Ruhe and MO Saliu. The art and science of software release planning. *IEEE Software* 2005; **22**, 47-53.
[22] S Weisberg. *Applied Linear Regression*. John Wiley and Sons, USA, 1980.
[23] J Xuan, H Jiang, Z Ren and Z Luo. Solving the large scale next release problem with a backbone based multilevel algorithm. *IEEE Trans. Software Eng.* 2012; **38**, 1195-212.