Int J Syst Assur Eng Manag (Jan-Mar 2015) 6(1):44–60
DOI 10.1007/s13198-014-0242-5

ORIGINAL ARTICLE

Bug prediction modeling using complexity of code changes

V. B. Singh · K. K. Chaturvedi · Sunil Kumar Khatri · Vijay Kumar

Received: 13 May 2013 / Published online: 7 March 2014

Abstract Researchers have proposed and implemented a plethora of bug prediction approaches in terms of different mathematical models for measuring the reliability growth of the software and to predict the latent bugs lying dormant in the software. During the last four decades, software reliability growth models (SRGM) have been successfully used to measure the reliability growth of closed source software. The SRGM developed were based on either calendar time or on testing effort. In late 90s, due to the advancement in communication and internet technologies, the development of open source software gets an edge and is proven to be very successful in different fields. Recently, researchers have measured the latent bugs in the open source software using an SRGM which has been developed for closed source software and concluded that the existing SRGM can well predict the latent bugs, but, still, it needs more investigation. In open source software, the source codes are frequently changes (the complexity of code changes) to meet the new feature introduction, feature enhancement and bug repair. In this paper, we have developed two complexity of code changes/entropy based bug prediction models namely (i) time vs entropy and (ii) entropy vs bugs. We have compared the proposed models with the existing time vs bugs SRGM. The empirical work has been carried out using three subsystems of Mozilla project. The statistical significance of different approaches has been tested using a non-parametric Kolmogorov–Smirnov (K–S) test. The bug prediction approaches have been compared on the basis of various performance measures namely R-Square (R^2), Adjusted R-Square (adj. R^2), Bias, variation and root mean square prediction errors. We found that the potential complexity of code changes based bug prediction approach i.e. time vs entropy is better over the time vs bugs and entropy vs bugs on the basis of different comparison criteria and statistical test.

Keywords Bug prediction · Entropy · Software reliability growth models · Complexity of code changes

V. B. Singh (✉)
Delhi College of Arts & Commerce, University of Delhi, Delhi, India
e-mail: vbsinghdcacdu@gmail.com

K. K. Chaturvedi
Department of Computer Science, University of Delhi, Delhi, India
e-mail: kkcchaturvedi@gmail.com

K. K. Chaturvedi
Indian Agricultural Statistics Research Institute (ICAR), New Delhi, India

S. K. Khatri
Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida, India
e-mail: sunilkkhatri@gmail.com

V. Kumar
Department of Mathematics, Amity School of Engineering and Technology, New Delhi, India
e-mail: vijay_parashar@yahoo.com

1 Introduction

In open source software, the requirements of the users are dynamic and large number of requests for bug fixes, feature enhancement and new feature introduction are reported through bug/issue reporting system. Software source codes are modified or enhanced to satisfy the user's need. The

enormous changes in the code make the software complex over a period of time. These changes increase the complexity of the code which also leads to the introduction of bugs. The changes are being made continuously in the software to remove the bugs, enhance the features and implement the new functionalities. The open source software/projects are being built through the contributors from diverse communities and keep on improving rapidly. The code complexity is one of the major attribute which determines the quality and reliability of the software. Many contributors are making changes in the source code of the software to produce the quality software in limited time. It is getting difficult to remember all those changes committed in the code which makes source code complex. The contributors are interacting with each other through the discussion forums/mailing lists etc. Three types of code changes occur in the source code namely, bug repair/bug fix, feature enhancement and addition of new features. Bugs are generated in the software mainly due to miscommunication or no communication among active users, frequently changing requirements, early release pressures, occurrence of programming errors, increasing software complexity, and bugs present in software development tools itself.

Entropy, an information theory based measure is defined as a measure of randomness/uncertainty/complexity in the code changes. This has been earlier attempted to quantify the code change and predict the bugs based on past defects using entropy (Hassan 2009). The software reliability growth models (SRGM) based on calendar time and testing efforts (time vs bugs) have been proposed in the literature and widely used in the industry (Goel and Okumoto 1979; Musa et al. 1987; Ohba 1984; Yamada et al. 1983; Singh et al. 2007; Pham 2006; Trivedi 2001; Xie 1991) .

In this paper, we have proposed models to determine the latent bugs lying dormant in the software by using two approaches namely, time vs entropy and entropy vs bugs. The proposed models have been developed in the line of existing SRGM. These approaches are summarized as follows:

Time vs bugs (reliability growth models)	In this approach, the reliability growth models namely expo nential (Goel and Okumoto 1979), delayed S-Shaped (Yamada et al. 1983), inflected delayed S-shaped (Ohba 1984) and power function (Singh et al. 2007) based models have been used to predict the potential bugs lying dormant in the software
Time vs entropy (potential complexity of code changes/ entropy based model)	Using this approach, we firstly predict the potential complexity of code changes/entropy and then it has been used to predict the potential bugs
Entropy vs bugs (complexity of code changes based models)	In this approach, we have proposed and developed the complexity of code changes based bug prediction models

To study the various bug prediction practices/approaches mentioned above, we have proposed the following research hypothesis:

Null hypothesis: The bug prediction approaches namely time vs bugs, time vs entropy and entropy vs bugs have no significant difference.

Alternate hypothesis: The bug prediction approaches namely time vs bugs, time vs entropy and entropy vs bugs have a significant difference.

The proposed models in the study have been applied to various components of Mozilla software to carry out the empirical analysis. The statistical significance has been tested using non parametric Kolmogorov–Smirnov (K–S) statistical test. The bug prediction scenario has been validated using R-squared (R^2), Adjusted R^2 (Adj. R^2), bias, variation and root mean square prediction errors (RMSPE).

The rest of the paper is divided into nine sections. Sect 2 describes the review of work available in the literature. Sect 3 defines the complexity of code changes. Sect 4 discusses bug prediction approaches and modelling. Sect 5 mentions the procedure of data collection and data preprocessing. In Sect 6, the results and discussions of the models are discussed. The managerial applications and threat to the validity has been discussed in Sects 7 and 8 respectively. Finally, the paper is concluded in Sect 9.

2 Review of work

A variety of approaches have been proposed in the literature for bug prediction such as software reliability growth models (Goel and Okumoto 1979; Huang et al. 1997; Kapur and Garg 1992; Kapur et al. 1999, 2008; Lyu 1996; Musa et al. 1987; Ohba 1984; Yamada et al. 1983), code metrics (lines of code) (Arisholm and Briand 2006; Gyimothy et al. 2005; Nagappan and Ball 2005a; Nagappan et al. 2006), process metrics (number of changes) (Hassan 2009; Nagappan and Ball 2005b) and previous defects (Hassan and Holt 2005; Kim et al. 2007; Ostrand et al. 2005). Reliability models have been developed to predict failure rates based on the expected operational usage

profile of the system. Defect detection rate during testing process was used to predict defects with an objective to reduce the software defects. The defects are minimized, if we incorporate the changes early in design and testing phases (Fenton and Neil 1999). The number of changes and the age of a module provide a good prediction over other product measures (Graves et al. 2000). The chaos in software and code development process has been studied in the literature and concluded that a complex process of coding had a negative effect of producing a complex system (Hassan and Holt 2003a, b). Researchers proposed in the literature that a prior modification to a file is a good predictor to determine the fault potential in the software (Arisholm and Briand 2006; Graves et al. 2000; Khoshgoftaar et al. 1999; Leszak et al. 2002). There is no single set of metrics which can predict bugs of multiple projects (Nagappan et al. 2006). A combination of product and process measures to predict the defect density using a decision tree algorithm was developed and validated with Mozilla releases. It is also concluded that the process measures are performing well over the product measures for bug prediction (Knab et al. 2006). A comparative study has been conducted for bug prediction using change metrics and code metrics for different releases of Eclipse and found that change metric gives better performance over the code metric (Moser et al. 2008). Change metrics achieve better prediction performance in the case of effort aware prediction models which consider the effort required for testing and code reviews (Kamei et al. 2010). History complexity metric (HCM) as a process measure has been proposed in the literature for bug prediction and compared with other approaches namely prior faults, prior modifications and complexity metrics. It is concluded that the proposed complexity metric is a better predictor of faults as compared to product measures (Hassan 2009). A benchmark study was conducted on defect prediction approaches and provided an extensive study to compare these models and metrics using decay based models (D'Ambros et al. 2010, 2012). A mathematical model was proposed to study the diffusion of the complexity of code changes in software and predicted the potential complexity of code changes (Chaturvedi et al. 2012). Recently, an attempt has been made to propose generalized decay models to measure the historical complexity metric with an intuition that the effect of code change can be reduced over a period (Singh and Chaturvedi 2012). Further, the authors applied support vector machine technique for regression and predicted the next year expected bugs based on the current year complexity of code changes/entropy (Singh and Chaturvedi 2013). To the best of our knowledge, no effort has been made to develop the complexity of code changes based mathematical models for predicting potential bugs. This paper investigates the applicability of the complexity of code changes based bug prediction. A comparison with the existing reliability growth models has been also proposed.

3 Complexity of code changes

The information theory deals with assessing and defining the amount of information contained in a message is measured as the amount of uncertainty or entropy of the distribution. The entropy H_n is defined by (Shannon 1948) as

$$H_n(P) = -\sum_{k=1}^{n} (P_k \log_2 P_k) \quad (1)$$

where $P_k \geq 0$ and $\sum_{k=1}^{n} P_k = 1$

where n is the number of files and the value of k varies from 1 to n. For a distribution P, where all the files have the same probability of changes $(P_k = 1/n; \forall\, k = 1, 2, \ldots, n)$, we achieve maximum entropy. On the other hand for a distribution P where a file i has a probability of code changes i.e., $P_i = 1$and $\forall k \neq i$, $P_k = 0$, we achieve a minimal entropy.

The entropy for this period can be calculated by substituting the values of these probabilities in Eq (1). From the definition, it is clear that the entropy will be maximum, if the changes are in every file. On the other hand, it will be minimum if the changes are occurring in a single file.

Code change process means to study the patterns of source code modifications. These modifications/changes are occurring due to bug repairs (BR), feature enhancement/modification (EM) and the addition of new features (NF). BR is the changes or modifications made in the code due to fixing of the bug. EM is the changes made due to some cosmetic changes like format, alignment, justification, comments etc. NF is the changes made in the code due to incorporation of new features or new components. These changes make the software code complex and may produce a system with new bugs. If the developer does not understand source code properly before performing changes/modifications in the source code, the release or new version of the software may get delayed. The frequent changes in the code may also negatively affect the software system in terms of its overall quality, reliability and sustainability.

The entropy based estimation may be helpful in studying the code change process. Code change quantifies the patterns of changes instead of simply measuring the number of changes to measure the effect of changes in code structure. This measurement is called the complexity of code changes. These changes can be studied at a source code level with granularity at package/file/class/method/variable at

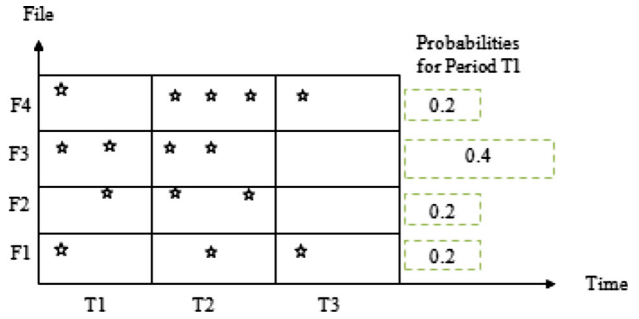

Fig. 1 Number of changes in files with respect to a specific period of time

the code level. These changes are measured at a specific duration which can be anything ranging from hours to year/decades depending on the frequency of updates/changes in the code. The complexity of code changes is calculated based on the number of changes in a file for specific periods. The period can be taken as a day, week, month, year etc. based on the total duration of the project or the number of changes occurs.

Hassan (2009) proposed and used entropy to measure the complexity of code changes. The probability P_k is defined as the ratio of the number of times kth file changed during a period and the total number of changes for all files in that period.

For example, suppose that there are 15 changes occurred in four files and three periods. These changes are shown in Fig. 1. For a first period T1, there are five changes occurred across all four files namely F1, F2, F3 and F4. There are eight changes in period T2 and two changes in period T3. All the four files are affected in period T2 and only two files are affected in period T3. The probability of these changes in files F1, F2, F3, and F4 during the period T1 will be 1/5 (=0.2), 1/5 (=0.2), 2/5 (=0.4) and 1/5 (=0.2) respectively. These probabilities have been shown in Fig. 1. By using these probabilities, we calculate the entropy and complexity of code changes for all the periods.

4 Bug prediction approaches and modelling

In this section, we firstly discussed the mathematical models used for measuring the reliability growth of software and after that on the line of these models, new models have been proposed to measure the potential complexity of code changes for bug prediction and to predict the bugs based on the complexity of code changes.

Notations

t	Time
X	Potential bugs
$m(t)$	Bugs occurred/removed up to time t
b	Bug detection/removal rate
β	Constant and >0
H_0	Potential complexity of code changes
$H(t)$	Complexity of code changes at time t
c	Rate of complexity of code changes
χ	Constant and >0
Z	Potential bugs due to the complexity of code changes
$m(H(t))$	Bugs occurred/diffused/removed up to $H(t)$
g	Rate of bug occurrence/diffusion in the software due to the code changes
δ	Constant and >0
k	Constant used in power function

4.1 Time vs bugs (reliability growth models)

In this approach, the reliability growth models namely GO (Goel and Okumoto 1979), Yamada delayed S-shaped (Yamada et al. 1983), inflected S-shaped (Ohba 1984), Kapur-Garg Model (Kapur and Garg 1992) and the instruction execution dependent models (Singh et al. 2007) have been used to predict the potential bugs.

There are several existing well-known non-homogenous poisson process (NHPP) models with different mean value functions. The mean value function $m(t)$ is described as the expected cumulative number of bugs in $(0, t]$ time interval. The SRGM are based on the following assumptions.

It has been observed that the relationship between the time and the corresponding number of bugs detected/removed is either exponential or S-shaped. Let $[N(t), t \geq 0]$ denote a discrete counting process representing the cumulative number of failures experienced (fault removed) up to time t, i.e., $N(t)$, is said to be a non-homogeneous poisson process (NHPP) with intensity function $\lambda(t)$, if it satisfies the following conditions:

There are no failures experienced at a time$t = 0$, i.e., $N(t = 0) = 0$ with probability 1.

The process has independent increments, i.e., the number of failures experienced in time interval, i.e., $N(t + \Delta t) - N(t)$, is independent of the history. Note this assumption implies the Markov property that $N(t + \Delta t)$ of the process depends only on the present state $N(t)$ and is independent of its past state$N(x)$, for $x < t$.

The probability that a failure will occur during $(t, t + \Delta t]$ is $\lambda(t)\Delta t + o(\Delta t)$, i.e., $\Pr[N(t + \Delta t) - N(t) = 1] = \lambda(t) + o(\Delta t)$. Note that the function $o(\Delta t)$ is defined as

$$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$$

In practice, it implies that the second or higher order effects of Δt are negligible.

The probability that more than one failure will occur during $(t, t + \Delta t]$ is $o(\Delta t)$, i.e., $\Pr[N(t + \Delta t) - N(t) > 1] = o(\Delta t)$.

The bug detection/removal process can be described as

$$\frac{dm(t)}{dt} \propto (X - m(t)) \text{ or } \frac{dm(t)}{dt} = b(t)(X - m(t)) \quad (2)$$

Here, $b(t)$is time dependent bug detection/removal rate.

Solving Eq (2) with initial condition, $m(0) = 0$ at a time $t = 0$, we get

$$m(t) = X\left(1 - e^{-\int_0^t b(t)\,dt}\right) \quad (3)$$

By putting different value of $b(t)$ in Eq (3) and solving with initial condition $m(0) = 0$, we get SRGM as follows in (Goel and Okumoto 1979; Ohba 1984; Yamada et al. 1983; Singh et al. 2007). These models for different values of $b(t)$ have been shown in Table 1.

In the above approach, the models have been developed by taking the calendar time i.e., t. These models have not been linked to the source code changes. In the following sections, we have proposed and developed models based on the source code changes i.e., the complexity of code changes.

4.2 Time vs entropy (potential entropy based model)

Using this approach, we firstly predict the potential complexity of code changes/entropy and after that the potential complexity of code changes has been used to predict the potential bugs lying dormant in the software.

The function $H(t)$ is called the mean value function and describes the expected complexity of code changes in $(0, t]$ time interval as described below by different mathematical models. The diffusion of the complexity of code changes in software has been described by the following equation:

$$\frac{dH(t)}{dt} \propto (H_0 - H(t)) \text{ or } \frac{dH(t)}{dt} = c(t)[H_0 - H(t)] \quad (4)$$

Here, $c(t)$ is rate at which the code are changes i.e., the complexity of code changes.

Solving the Eq (4) with initial condition, $H(0) = 0$ at a time $t = 0$, we get

$$\Rightarrow H(t) = H_0\left(1 - e^{-\int_o^t c(t)dt}\right) \quad (5)$$

We take different value of $c(t)$ to derive different models which are shown in Table 2.

We calculate the potential complexity of code changes diffused in the software using proposed models i.e., Model 7 to Model 12 which are shown in Table 2. After getting the potential complexity of code changes, we apply simple linear regression to find the potential bugs diffused in the software.

The simple linear regression (Weisberg 1980) has been fitted with the complexity of code changes as independent variable $H(t)$ and bugs detected as dependent variable $m(t)$ using an equation

$$m(t) = \beta_0 + \beta_1 * H(t) \quad (6)$$

where β_0 and β_1 are regression coefficients

After getting the regression coefficients β_0 and β_1, we obtain the potential bugs X for potential entropy H_0 in the software by putting the value of the potential complexity of code changes obtained from the above proposed models shown in Table 2.

Table 1 Time vs bugs (existing software reliability growth models)

Mean value function, $m(t) = X\left(1 - e^{-\int_0^t b(t)\,dt}\right)$

$b(t) = b$	**Model 1**: $m(t) = X(1 - e^{-bt})$	Reference (Goel and Okumoto 1979)
$b(t) = \frac{b^2 t}{1+bt}$	**Model 2**: $m(t) = X(1 - (1 + bt)e^{-bt})$	Reference (Yamada et al. 1983)
$b(t) = \frac{b}{(1+\beta e^{-bt})}$	**Model 3**: $m(t) = X\frac{(1-e^{-bt})}{(1+\beta e^{-bt})}$	Reference (Kapur and Garg 1992; Ohba 1984)
$b(t) = bt^k$	**Model 4**: $m(t) = X\left[1 - e^{-\frac{bt^{k+1}}{k+1}}\right]$	Reference (Singh et al. 2007)
$b(t) = \frac{b^2 t^{k+1}}{1+bt^{k+1}}$	**Model 5**: $m(t) = X\left[1 - \left(1 + b\frac{t^{k+1}}{k+1}\right)e^{-\left(\frac{bt^{k+1}}{k+1}\right)}\right]$	Reference (Singh et al. 2007)
$b(t) = \frac{bt^k}{1+\beta e^{-\left(bt^{k+1}k+1\right)}}$	**Model 6**: $m(t) = X\left[\frac{1-e^{-\left(\frac{bt^{k+1}}{k+1}\right)}}{1+\beta e^{-\left(\frac{bt^{k+1}}{k+1}\right)}}\right]$	Reference (Singh et al. 2007)

If $k = 0$, model 4, model 5 and model 6 reduces to model 1, model 2 and model 3 respectively

Table 2 Time vs entropy (proposed potential complexity of code changes based models)

Mean value function, $\mathrm{H}(t) = H_0\left(1 - \mathrm{e}^{-\int_o^t c(t)dt}\right)$	
$c(t) = c$	**Model 7**: $H(t) = H_0\,(1 - e^{-ct})$
$c(t) = \frac{c^2 t}{1+ct}$	**Model 8**: $H(t) = H_0\,(1 - (1 + ct)e^{-ct})$
$c(t) = \frac{c}{(1+\chi e^{-ct})}$	**Model 9**: $H(t) = H_0 \frac{(1-e^{-ct})}{(1+\chi e^{-ct})}$
$c(t) = ct^k$	**Model 10**: $H(t) = H_0\left(1 - e^{-\frac{ct^{k+1}}{k+1}}\right)$
$c(t) = \frac{\left(c^2\frac{t^{k+1}}{k+1}\right)}{\left(1+c\frac{t^{k+1}}{k+1}\right)}$	**Model 11**: $H(t) = H_0\left[1 - \left(1 + c\,\frac{t^{k+1}}{k+1}\right)e^{-\left(\frac{ct^{k+1}}{k+1}\right)}\right]$
$c(t) = \frac{ct^k}{\left(1+\chi e^{-\left(\frac{ct^{k+1}}{k+1}\right)}\right)}$	**Model 12**: $H(t) = H_0\left[\frac{\left(1-e^{-\left(\frac{ct^{k+1}}{k+1}\right)}\right)}{\left(1+\chi e^{-\left(\frac{ct^{k+1}}{k+1}\right)}\right)}\right]$

If $k = 0$, model 10, model 11 and model 12 reduces to model 7, model 8 and model 9 respectively

4.3 Entropy vs bugs (complexity of code changes)

In this approach, we have developed the complexity of code changes based models for bug prediction.

The function $m(H(t))$ is the mean value function and describes the expected cumulative potential complexity of code changes in (0, $H(t)$) interval. The bug diffusion/detection process can defined as

$$\frac{\mathrm{d}m(t)/\mathrm{d}t}{\mathrm{d}H(t)/\mathrm{d}t} \propto [Z - m(H(t))] \text{ or } \frac{\mathrm{d}m(t)/\mathrm{d}t}{\mathrm{d}H(t)/\mathrm{d}t} = g(H(t))(Z - m(H(t))) \tag{7}$$

$g(H(t))$ is the rate of bug occurrence/diffusion in the software due to the complexity of code changes at any given time t.

With the initial conditions $H(0) = 0$ at the time $t = 0$, by solving the above equation, we get

$$m(t) = Z\left(1 - e^{-\int_0^t g(H(t))d(H(t))}\right) \tag{8}$$

If we take different values of $g(H(t))$ in Eq (8), we get different proposed models based on these values as shown in Table 3.

The existing and proposed models are shown in Tables 1, 2 and 3. The unknown parameters of the models have been estimated using Statistical Package for Social Sciences (SPSS) software.

5 Data collection and preprocessing

We have taken the complexity of code changes of "mozilla/layout/svg/", "mozilla/layout/base/", and "mozilla/layout/xul/" of the Mozilla project (The Mozilla project 2013) and the bugs which have been reported in BugZilla (The bugZilla project 2013) bug reporting and tracking system. The process

Table 3 Entropy vs bugs (proposed complexity of code changes based models)

Mean value function, $m(t) = Z\left(1 - e^{-\int_0^t g(H(t))d(H(t))}\right)$	
$g(\mathrm{H}(t)) = g$	**Model 13**: $m(\mathrm{H}(t)) = Z(1 - e^{-gH(t)})$
$g(\mathrm{H}(t)) = \frac{g^2 H(t)}{1+gH(t)}$	**Model 14**: $m(\mathrm{H}(t)) = Z(1 - (1 + g\mathrm{H}(t))e^{-gH(t)})$
$g(\mathrm{H}(t)) = \frac{g}{(1+\delta e^{-gH(t)})}$	**Model 15**: $m(\mathrm{H}(t)) = Z\frac{\left(1-e^{-gH(t)}\right)}{(1+\delta e^{-gH(t)})}$
$g(\mathrm{H}(t)) = gH(t)^k$	**Model 16**: $m(\mathrm{H}(t)) = Z\left[1 - e^{-\left(\frac{gH(t)^{k+1}}{k+1}\right)}\right]$
$g(\mathrm{H}(t)) = \frac{\left(g^2\frac{H(t)^{k+1}}{k+1}\right)}{\left(1+g\frac{H(t)^{k+1}}{k+1}\right)}$	**Model 17**: $m(\mathrm{H}(t)) = Z\left[1 - \left(1 + g\,\frac{\mathrm{H}(t)^{k+1}}{k+1}\right)e^{-\left(\frac{gH(t)^{k+1}}{k+1}\right)}\right]$
$g(\mathrm{H}(t)) = \left(\frac{gH(t)^k}{1+\delta e^{-\left(\frac{gH(t)^{k+1}}{k+1}\right)}}\right)$	**Model 18**: $m(\mathrm{H}(t)) = Z\left(\frac{1-e^{-\left(\frac{gH(t)^{k+1}}{k+1}\right)}}{1+\delta e^{-\left(\frac{gH(t)^{k+1}}{k+1}\right)}}\right)$

If $k = 0$, model 16, model 17 and model 18 reduces to model 13, model 14 and model 15 respectively

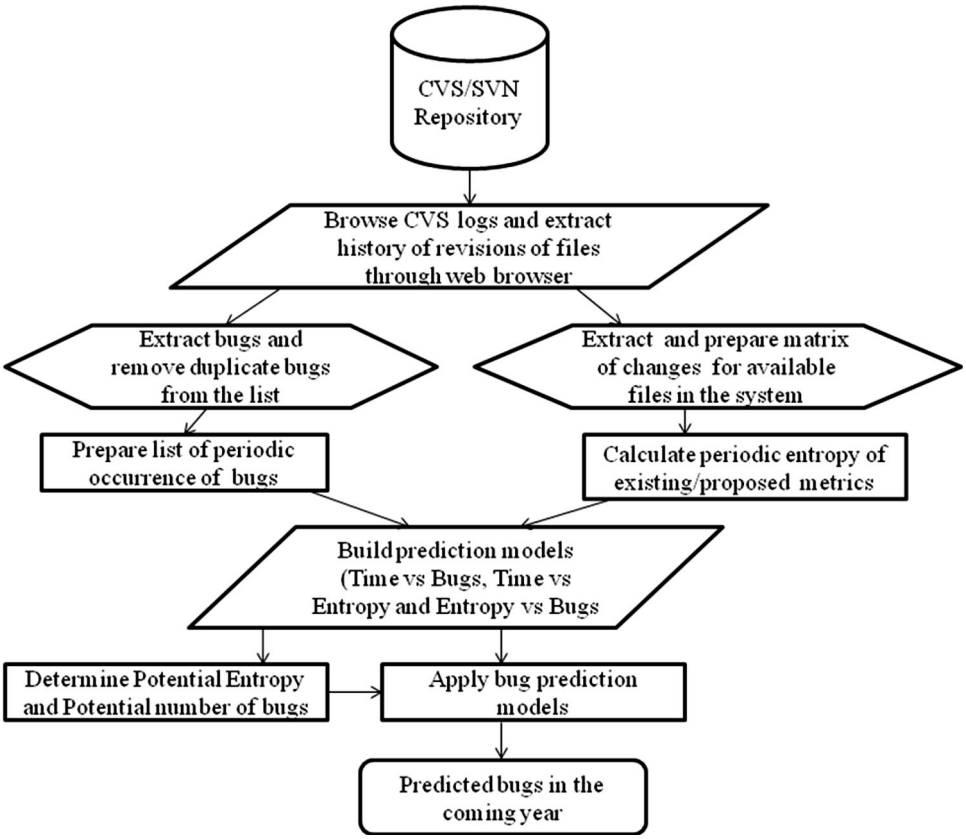

Fig. 2 Process for building the entropy based prediction models

of data collection, extraction and model building are shown in Fig. 2. These subsystems are selected as test cases and historical changes of the files of these subsystems are extracted from the concurrent versioning system (CVS) logs repository. After extracting data from the repository, monthly number of changes and the number of bugs occurred/fixed during that period have been recorded for all the files.

These bugs are arranged based on their first appearance for a fix and remaining duplicate entries of these bugs are discarded. In this study, we have considered the period as the calendar months of a year. There are 52 files in the subsystem "mozilla/layout/svg/", 60 files in the subsystem "mozilla/layout/base/", and 106 files in the subsystem "mozilla/layout/xul/". The methodology for data collection and modelling are as follows:

Step 1 Choose the project
Step 2 Select the sub-systems
Step 3 Browse the CVS logs for historical changes
a. Arrange the files as per their commit/transaction date of the changes
b. Count month wise changes with respect to various files in the chosen sub-system
c. Calculate entropy/complexity of code changes of the files
Step 4 Browse the CVS logs for historical changes
a. Extract the bugs detected/removed during the period under consideration
b. Arrange the bugs as per their arrival or first appearance and remove other duplicate occurrences
c. Count month wise bugs of the chosen sub-system
Step 5 Build the Prediction Models using different approaches
Step 6 Determine the potential complexity of code changes and the potential number of bugs in the software
Step 7 Apply the K–S test for statistical significance using data in step6
Step 8 Calculate the entropy of the current period and predict the number of bugs for the coming year

These detected/removed bugs are extracted from the CVS log repository and confirms with the bug reporting system. A sample of CVS logs in a particular file of the specific subsystem is also shown in Fig. 3.

In Fig. 3, date wise changes are recorded for a specific file nsBidiPresUtils.h with a specific bug in the log entry.

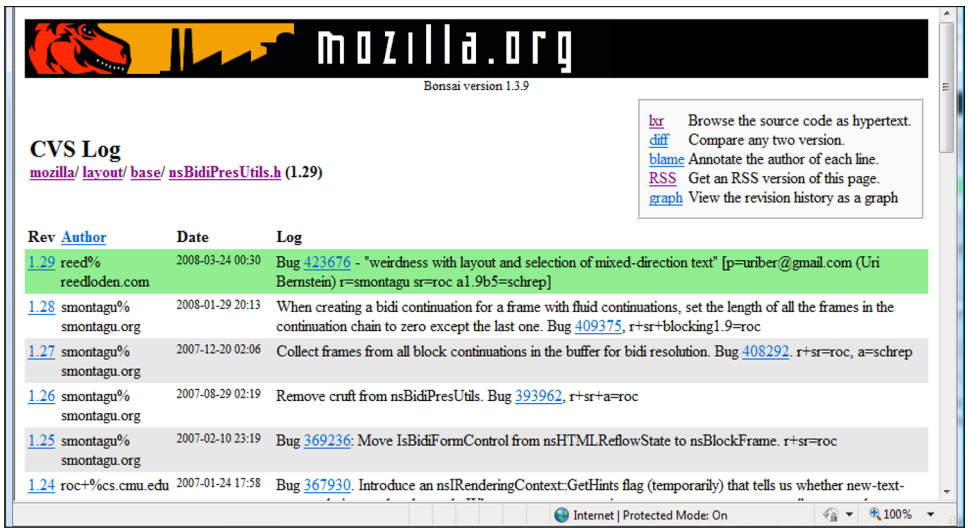

Fig. 3 CVS log output of the subsystem of Mozilla (Source: http://bonsai.mozilla.org/cvslog.cgi?file=mozilla/layout/base/nsBidiPresUtils.h&rev=HEAD&mark=1.29)

Prediction Error (Pred. Error)

It is the difference between observed value and predicted value. The lower value of Prediction error indicates less fitting error, thus indicates better goodness of fit

$$\Pr ed.Error = (ObservedValue - \Pr\ edictedValue)$$

Bias

It is the average of prediction error. Lower the value of bias, provides better goodness of fit

$$Bias = \frac{\sum (Observed - Predicted)}{n}$$

Variation

The standard deviation of prediction error is known as variation. Lower value of variation indicates better goodness of fit

$$Variation = \sqrt{\frac{\sum (\Pr\ ed.Error - Bias) * (\Pr\ ed.Error - Bias)}{n-1}}$$

Coefficient of determination (R-square)

This coefficient is defined as the ratio of the sum of squares between corrected and residuals subtracted from 1. This measure is used to test the significant trend between predicted and observed values.

$$R^2 = 1 - \frac{CorrectedSS}{\mathrm{Re}\ sidualSS}$$

R^2 measures the total variation about the mean for the fitted curve. It ranges in value from 0 to 1. Lower values indicate that the model does not fit the data well and the larger R^2 indicates the better fit of the model and able to explain the variation in data

Root mean squared prediction error (RMSPE)

It is a measure of closeness with which a model predicts the observation. Lower value of RMSPE provides better goodness of fit

$$RMSPE = \sqrt{(Bias * Bias + Variation * Variation)}$$

Kolmogorov–Smirnov (K–S) test

The K–S test is a non-parametric test and it is free from the distribution. It does not consider any assumption about the data. This test is able to provide the distance between the observed and predicted curve with the confidence level in terms of proba bility.Lower the value of distance and higher the value of probability provides the signifi cantly improved performance of the model

The trends of complexity of code changes/entropy and bugs for the considered subsystem under study are shown in Figs. 4, 5 and 6 for subsystems SVG, Base and XUL respectively. It is clear from these figures that the entropy and bugs fixed/occurred are highly correlated.

6 Results and discussions

The data has been collected for three subsystems of layout components of Mozilla namely SVG, Base, and XUL. The entropy/complexity of code changes is calculated on a

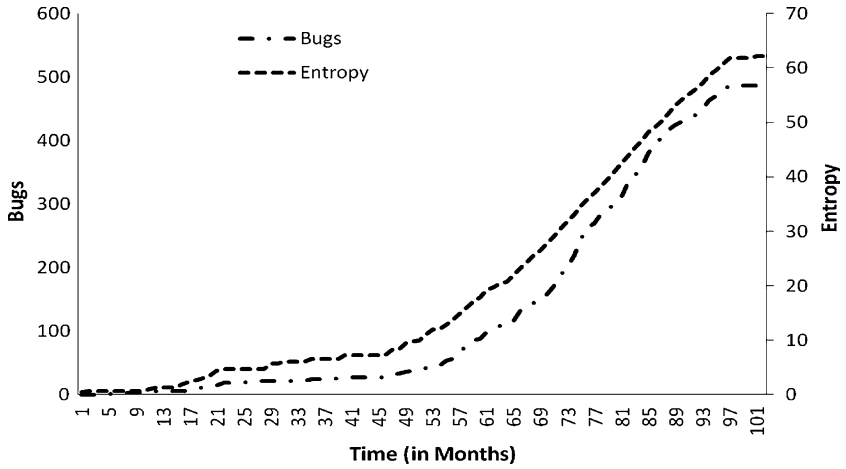

Fig. 4 Entropy and bug occurrence trends of SVG dataset

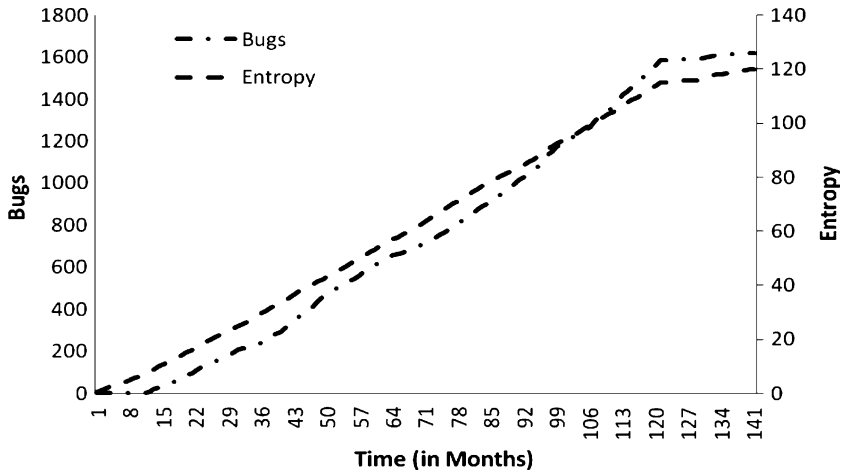

Fig. 5 Entropy and bug occurrence trends of base dataset

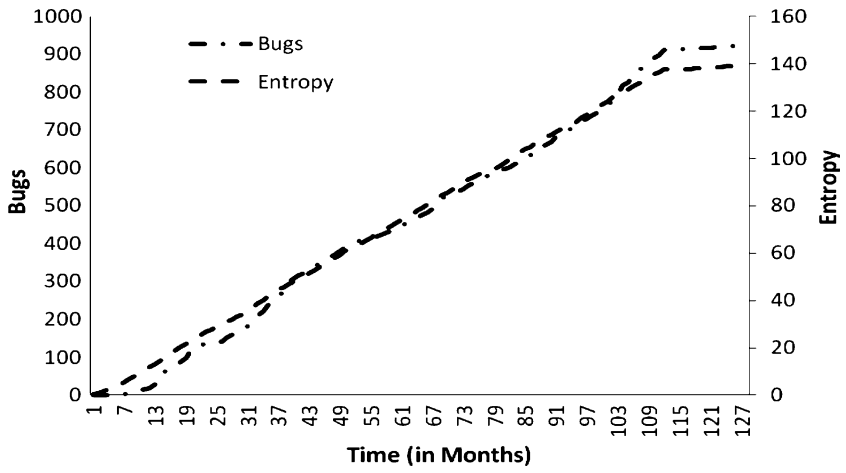

Fig. 6 Entropy and bug occurrence trends of XUL dataset

monthly basis with respect to changes in the files of these subsystems. The parameters of these models have been estimated by applying the non-linear regression in SPSS. Simple linear regression has been applied between the complexity of code changes and bugs to obtain the regression coefficients. These regression coefficients have been used to calculate the potential bugs based on the potential complexity of code changes estimated using entropy based models. The statistical performance and regression coefficients for the considered data sets are shown in Table 4. The values of R^2 are found more than 98 % in all cases. The standard error of the estimates has been found to be 19.0108, 48.8429 and 20.6138 for SVG, Base and XUL respectively which shows the appropriateness of the model fitting.

We have estimated the parameters of different proposed models with respect to all the cases i.e., namely software reliability growth models, the potential complexity of code changes based models and the complexity of code changes based models using the SPSS package.

The parameters of SRGM based models developed in literature i.e., from Model 1 to Model 6 are estimated using nonlinear regression and the results of this estimation is shown in Table 5. The value of R^2 is more than 0.99 for all the models across all the datasets except models mentioned Model 1 and Model 2 for SVG dataset. The bug detection/removal rate i.e., b is varied from 0.001 to 0.095 for SVG, 0.001–0.013 for Base and 0.001–0.021 for XUL datasets. Model 1 does not give the accurate value of the parameters as this model is exponential in nature and does not fit on SVG dataset. Estimated potential bugs are shown in Table 5 in the third column as X. The potential bugs for the Model 1 is in the range of 195,400, 12,450 and 9,845 for datasets SVG, Base and XUL respectively. Model 1 is not fitted well as shown by the value of R^2 for the SVG dataset. The value of b is also much less for this dataset i.e., 1.71E−05. If we further look into the table, we found that the values of X for the Model 2 are also very large i.e., 1,532 with the R^2 0.845. In the rest of the other cases shown in Table 5, the value of R^2 is more than 0.99 and the potential bugs are within the range.

Table 4 Statistical performance parameters of entropy vs bugs using simple linear regression

Data sets	Regression parameters		R^2	Adjusted R^2	Std. error of the estimate
	β_0	β_1			
SVG	−27.9694	8.1371	0.9875	0.9873	19.0108
Base	−144.8785	14.4216	0.9923	0.9923	48.8429
XUL	−41.5977	6.7249	0.9953	0.9953	20.6138

The goodness of fit curve has been shown in Figs. 7 and 8 for the models which give the maximum fit on the basis of R^2 for given datasets. The results are confirmed by drawing the goodness of the fit curve for the Model 3 and Model 6 in the Figs. 7 and 8 respectively. Model 3 fits well with the predicted bugs and the actual bugs lying/detected in the software. Model 6 shows the large variation for SVG and Base dataset in the later part of the software maintenance. Larger value of β shows the poor maintenance (bug removal is delayed) in SVG as compared to smaller value of β for XUL dataset. The value of k is found significant in all the models namely Model 4, Model 5 and Model 6 using power functions. For management decision we observed that the predicted potential bugs using Model 3, 4, 5 and 6 are closer to the detected bugs i.e., 488 for SVG data set. The difference in R^2 value of these models is also non-significant. For Base data set, the Model 3 and Model 6 are estimated potential bugs closer to the detected bugs. These models are also showing the improved performance in terms of R^2 which is significant. For XUL data set, Model 2, Model 3 and Model 5 shows the significantly closer potential bugs with the detected bugs in the component i.e., 922. But the R^2 is improved only in case of Model 4 and Model 6 which have a significantly higher value of potential bugs.

The parameters of the potential complexity of code changes based models proposed in Model 7 to Model 12

Table 5 Parameter estimates for different models of time vs bugs (software reliability growth models)

Data sets	Models	Parameter estimates				
		X	b	β	k	R^2
SVG	Model 1	195,400	1.71E−05	–	–	0.677
	Model 2	1,532	0.01	–	–	0.845
	Model 3	558	0.095	17.14	–	0.997
	Model 4	515	9.28E−11	–	–	0.996
	Model 5	541	1.41E−06	–	2.507	0.995
	Model 6	515	0.001	32.22	1.295	0.998
Base	Model 1	12,450	0.001	–	–	0.953
	Model 2	3,179	0.013	–	–	0.998
	Model 3	2,032	0.03	12.241	–	0.996
	Model 4	2,377	0.001	–	0.792	0.996
	Model 5	2,567	0.011	–	0.098	0.995
	Model 6	2,021	0.005	3.554	0.349	0.996
XUL	Model 1	9,845	0.001	–	–	0.987
	Model 2	1,255	0.021	–	–	0.991
	Model 3	1,330	0.021	4.00	–	0.993
	Model 4	1,709	0.002	–	0.346	0.994
	Model 5	1,255	0.021	–	0	0.991
	Model 6	1,418	0.005	1.102	0.228	0.994

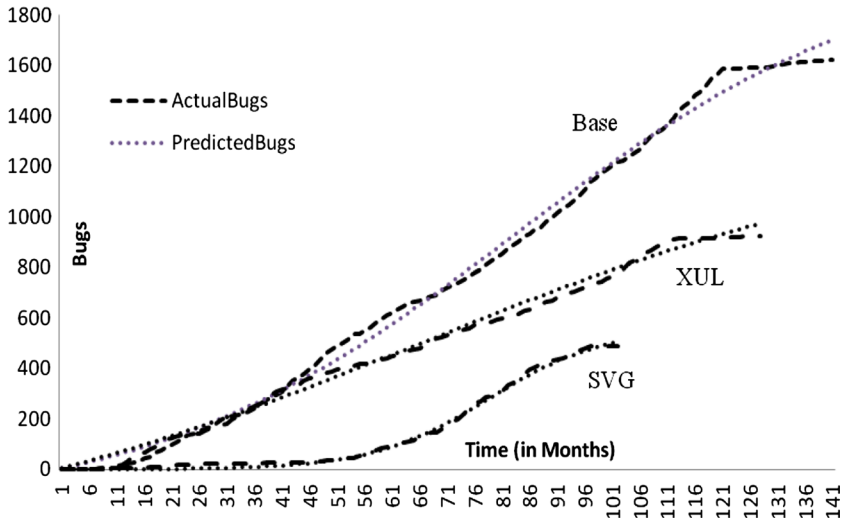

Fig. 7 Goodness of fit for model defined in Model 3

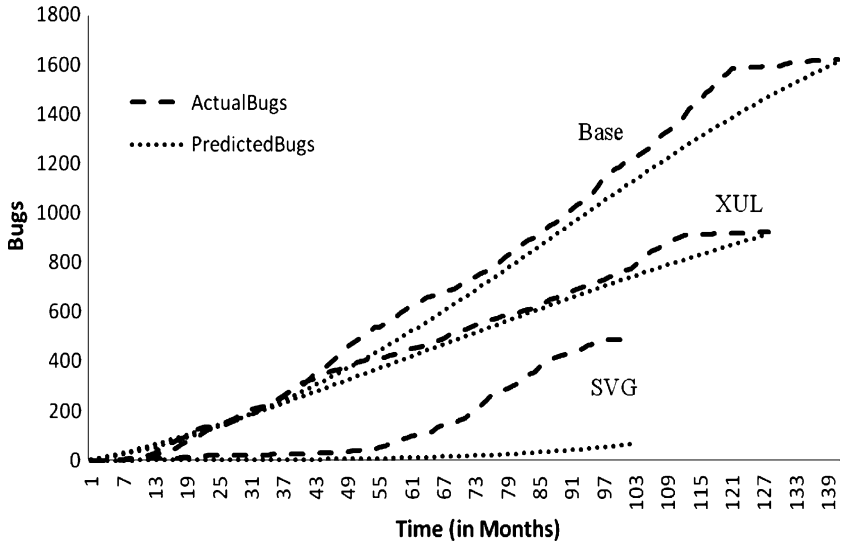

Fig. 8 Goodness of fit for model defined in Model 6

has been estimated using nonlinear regression and the results are shown in Table 6. The value of R^2 is more than 0.99 for all the models across all the datasets except model discussed in Model 8 and Model 9 for SVG dataset. For model mentioned in the Model 8 does not give the accurate value of the parameters as this model is exponential and does not fit on SVG dataset. This potential entropy or potential complexity of code changes will help in predicting the future potential bugs lying dormant in the software using simple linear regression. These regression parameters shown in Table 4 are used with the potential complexity of code changes to determine the potential bugs lying dormant in the software. These potential bugs are shown as the last column of the Table 6. The goodness of fit curve has been shown in Figs. 9 and 10 for the models which give the maximum fit on the basis of R^2.

From the Table 6, it is also clear that the potential bugs with respect to Model 7 are found 246,933, 16,209 and 6,298 for SVG, Base and XUL respectively. The value of R^2 for Model 7 and Model 8 are 0.784 and 0.971 for SVG dataset. The value of R^2 is more than 0.99 in the rest of the other models and datasets. The rate of complexity of code changes is also in the range of 0.001–0.06 for all the proposed models based on the complexity of code changes. The potential bugs predicted using all these proposed models are at par with the actual bugs except Model 7. Model 8 which is overestimated and also showing the poor value of R^2 i.e., 0.971 for SVG dataset.

For SVG data set, the value of potential bugs for Model 9, Model 10 and Model 12 give fair value against the bugs detected i.e., 488. This shows that the remaining bugs still need to be removed from the software. Since, Model 9 and Model 12 gives better R^2, we will trust these models for determining remaining bugs yet to be removed from the software. For Base data set, the potential bugs are estimated at par i.e., 1,621 for the Model 8, Model 9, Model 11

Table 6 Parameter estimates for different models for time vs entropy (potential complexity based models)

Data sets	Models	Parameter estimates					Potential bugs
		H_0	c	χ	k	R^2	
SVG	Model 7	3.04E + 04	1.54E−05	–	–	0.784	246,933
	Model 8	3.51E + 03	0.02	–	–	0.971	28,525
	Model 9	83.985	0.06	125.665	–	0.996	655
	Model 10	91.134	1.28E−06	–	2.245	0.992	714
	Model 11	128.049	0.001	–	0.786	0.991	1,014
	Model 12	77.404	0.009	31.385	0.443	0.996	602
Base	Model 7	1,134	0.001	–	–	0.994	16,209
	Model 8	156.02	0.02	–	–	0.995	2,105
	Model 9	153.02	0.022	4.3	–	0.999	2,062
	Model 10	193.376	0.002	–	0.354	0.998	2,644
	Model 11	156.02	0.02	–	0	0.995	2,105
	Model 12	153.02	0.022	4.3	0	0.999	2,062
XUL	Model 7	942.694	0.001	–	–	0.995	6,298
	Model 8	171.661	0.025	–	–	0.993	1,113
	Model 9	181.883	0.022	3.121	–	0.998	1,182
	Model 10	226.636	0.003	–	0.272	0.998	1,483
	Model 11	171.661	0.025	–	0	0.993	1,113
	Model 12	181.883	0.022	3.121	2.18E − 08	0.998	1,182

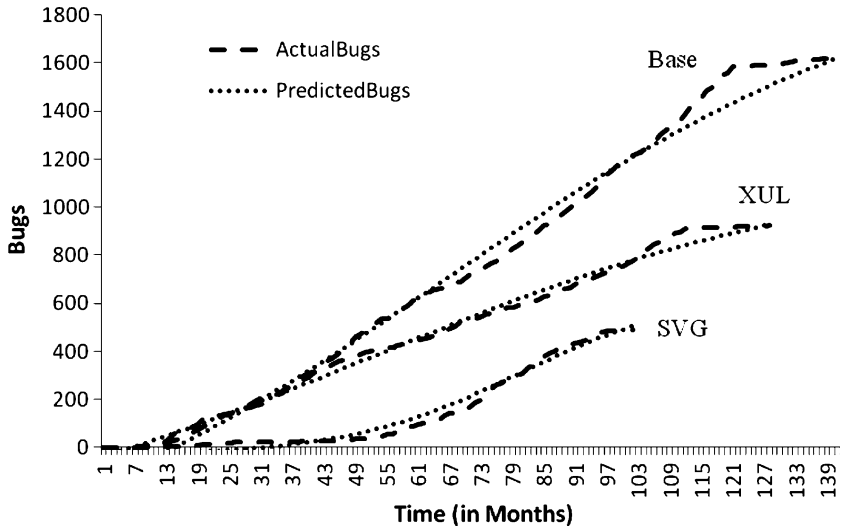

Fig. 9 Goodness of fit for model defined in Model 9

and Model 12. By the comparison between these models based on value of R^2, it is found that for Model 9 and Model 12 with 0.999 and 0.998 are slightly improved over other models. Similar trends have also been observed in the XUL data set with the potential bugs are estimated at 922 which is closer to the predicted potential bugs lying in the software. The Model 9 and Model 12 show the improved performance over other models. The goodness of fit curves is shown in Figs. 9 and 10 for Model 9 and Model 12 respectively. The curve with respect to SVG dataset shows the best fit using both these models.

The parameters related to complexity of code changes with respect to bugs detected/removal for Model 13 to Model 18 has been estimated using nonlinear regression and the result is shown in Table 7. The value of R^2 is more than 0.98 for all the models and for all the datasets except Model 13 for SVG and Base dataset. Model 13 does not give the accurate value of the parameters as this model is exponential and does not fit on SVG and Base datasets. The values of potential bugs detected/removed due to diffusion of complexity of code changes are in the range of 5.88E−05 to 0.06. The value of potential bugs using the diffused

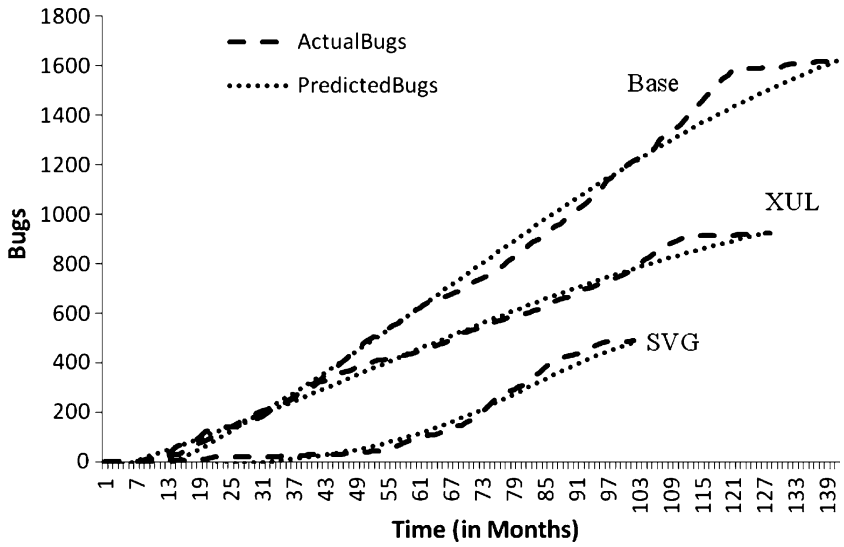

Fig. 10 Goodness of fit for model defined in Model 12

Table 7 Parameter estimates for different models of various entropy vs bugs (complexity of code changes)

Data sets	Models	Parameter estimates				
		Z	*g*	Δ	*k*	R^2
SVG	Model 13	126,900	5.88E−05	–	–	0.974
	Model 14	920	0.029	–	–	0.998
	Mode1 15	598	0.066	12.243	–	0.999
	Model 16	855	0.0,016	–	0.6389	0.998
	Model 17	918	0.0288	–	0.0010	0.998
	Model 18	600	0.0188	4.0382	0.3028	0.999
Base	Model 13	13,770	0.0013	–	–	0.968
	Model 14	3,137	0.014	–	–	0.996
	Mode115	3,338	0.018	6.965	–	0.998
	Model 16	1,800	0.009	–	0.062	0.889
	Model 17	2,467	0.013	–	0.093	0.993
	Model 18	3,429	0.01	4.534	0.101	0.998
XUL	Model 13	5,771	0.001	–	–	0.985
	Model 14	1,310	0.017	–	–	0.987
	Mode1 15	1,399	0.017	–	–	0.993
	Model 16	2,412	0.001	–	0.277	0.995
	Model 17	1,310	0.017	–	0	0.987
	Model 18	1,000	0.009	2.823	0.233	0.985

complexity of code changes using Model 13 i.e., *Z*are 126,900, 13,770 and 5,771 for SVG, Base and XUL dataset respectively. The value of R^2 is also very low for these datasets using Model 13 i.e., 0.974, 0.968, and 0.985 for SVG, Base and XUL dataset respectively. For SVG data set, the predicted potential bugs for Model 15 and 18 are closer to the detected bugs i.e., 488. The value of R^2 as 0.999 also confirms the fitting of these models for this data set using these proposed models. For Base data set, the value of predicted potential bugs estimated using the model 16 is 1,800 which is closer to detect bugs. In this model, the R^2 is least i.e., 0.889 for Model 16 as compared to other models. Model 15 and Model 18 predict the fair amount of potential bugs lying in the software components as the larger is the value of R^2 for these models. This confirms with the bug arrival/detected patterns of the software components because of the sharp increase after 105 months. For XUL data set, the closer value with the predicted potential bugs is found for the model 14, 15, 17 and 18 but the R^2 is poor as compared to Model 16. This is

due to sudden increase in the arrival/detection of bugs in the software.

The goodness of fit curve has been shown in Figs. 11 and 12 for the models which give the maximum fit on the basis of R^2 for given datasets. The goodness of fit curve of the Model 18 for Base dataset is shown best fitting. In the rest of the cases, the performance diminishes.

To summarize the performance of the proposed models based on goodness of fit criteria i.e., R^2, we have found that there are 41 cases out of 54 cases (6 models, 3 data sets and 3 approaches) which shows the value of R^2 is more than 99 %. It is also observed that 14 of software reliability growth models based (time vs bugs) approach, 16 of potential complexity of code changes based (time vs entropy) approach and 11 cases of complexity of code changes based (entropy vs bugs) approach out of 18 cases for each category show the accuracy more than 99 %.

We further investigated the model which gives the best performance in three approaches on the basis of Bias, variation and RMSPE (root mean squared prediction error).

The statistical performances of the best models obtained in the above proposed models are shown in the Table 8. In this table, the value of R^2 is more than 98 % which shows all models are applicable in determining the potential bugs in the software project based on complexity of code change metric. If we compare these models based on other performance measures namely, Bias, variation and RMSPE, it is found that the time vs entropy based models are having lower values of these performance measures. Further, we have confirmed it with the non-parametric Kolmogorov–Smirnov (K–S) test.

Further, we have applied the Kolmogorov–Smirnov test (K–S test) to statistically validate the proposed models by considering the actual and predicted bugs/complexity of

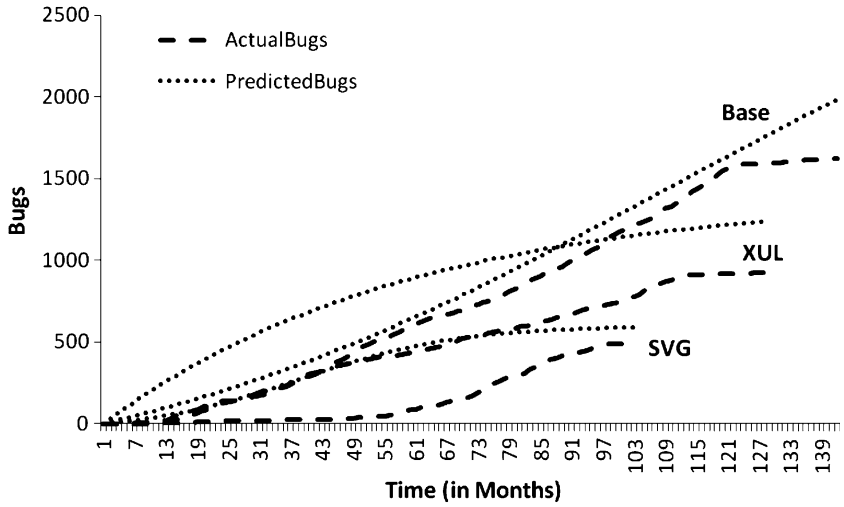

Fig. 11 Goodness of fit for model defined in Model 15

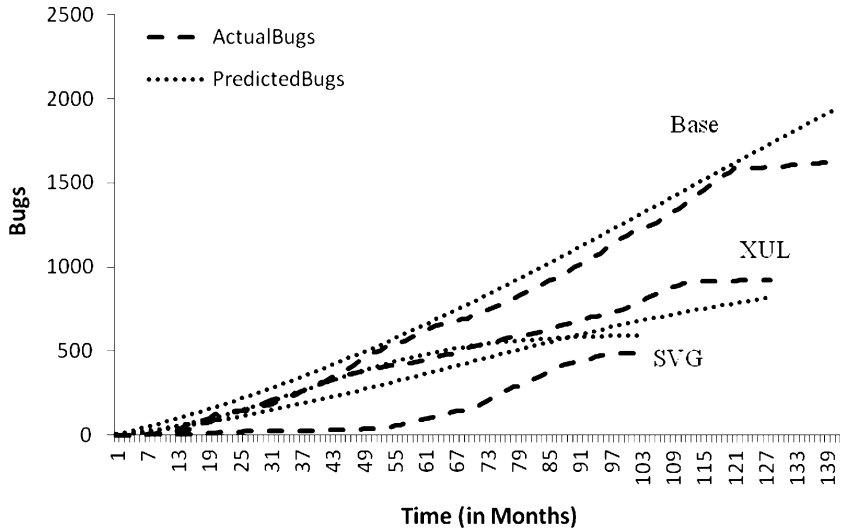

Fig. 12 Goodness of fit for model defined in Model 18

Table 8 Statistical performance parameters of best models of various datasets

Cases	Models	Data sets	Statistical performance parameters			
			Bias	Variation	RMSPE	R^2
Time vs bugs	Model 3	SVG	5.552	8.186	9.892	0.997
		Base	−5.127	36.808	37.163	0.998
		XUL	−10.393	23.802	25.972	0.996
	Model 6	SVG	128.151	151.715	198.595	0.996
		Base	60.190	55.341	81.765	0.993
		XUL	26.979	30.429	40.667	0.994
Time vs entropy	Model 9	SVG	0.278	1.307	1.336	0.996
		Base	1.034	1.336	1.690	0.996
		XUL	0.811	1.894	2.060	0.999
	Model 12	SVG	123.479	149.183	193.656	0.999
		Base	712.936	518.631	881.621	0.998
		XUL	400.549	256.671	475.731	0.998
Entropy vs bugs	Model 15	SVG	−211.472	126.563	246.452	0.999
		Base	−107.875	63.564	125.209	0.999
		XUL	−357.048	95.219	369.527	0.998
	Model 18	SVG	−211.248	125.864	245.901	0.998
		Base	−89.315	56.185	105.517	0.993
		XUL	79.163	48.494	92.836	0.985

code changes of the system. Kolmogorov–Smirnov (K–S) statistic quantifies a distance between the observed value and the predicted value of the cumulative distribution function for the given distribution. The K–S test has been applied to the models selected based on the goodness of fit criteria. The results of the K–S test are shown in Table 9. We have applied K–S test on the models which give the best performance on the basis of R^2, Bias, Variation, and RMSPE (the results are shown in Tables 4, 5, 6, 7, 8).

From Table 9, we have selected Model 3 and Model 6 for time vs bugs, Model 9 and Model 12 for time vs entropy and Model 15 and Model 18 for entropy vs bugs approaches. The D-value shows the distance between the actual and predicted values. The high value of performance parameter of associated *P* value shows the applicability of that model. It is clear from the above table that for time vs bugs approach; Model 3 gives associated *P* value 0.7677 and 0.8992 for the Base and XUL data sets. In time vs entropy approach, Model 9 gives associated *P* value 0.9583 and 0.9733 for XUL and Base data sets. For this approach also, Model 12 gives the associated *P* value 0.9583 and 0.9733 for XUL and Base data sets. The associate *P* value in entropy vs bugs approach is in the range of 0.00–0.1070 which is not significant. If we compared across different types of cases for the SVG dataset, we found that the maximum associated *P* value of time vs entropy case is 0.2713. This value is non significant. This result in rejecting the Null Hypothesis i.e., the alternate hypothesis is accepted. This analytical results show that the time vs entropy based approaches show significantly improved performance over the other two approaches.

Table 9 Statistical performance using Kolmogorov–Smirnov (K–S) test of different models

Cases	Models	Data sets	Kolmogorov–Smirnov	
			D-value	Associated *P* value
Time vs bugs	Model 3	SVG	0.2059	0.0225
		Base	−0.0780	0.7677
		XUL	−0.0703	0.8992
	Model 6	SVG	0.4902	0.0000
		Base	0.1277	0.1866
		XUL	0.125	0.2525
Time vs entropy	Model 9	SVG	0.1373	0.2713
		Base	0.5670	0.9733
		XUL	0.0625	0.9583
	Model 12	SVG	0.1471	0.2023
		Base	0.5670	0.9733
		XUL	0.0625	0.9583
Entropy vs bugs	Model 15	SVG	−0.4314	0.0000
		Base	−0.1560	0.0579
		XUL	−0.5078	0.0000
	Model 18	SVG	−0.4314	0.0000
		Base	−0.1418	0.1070
		XUL	0.1953	0.0129

From above experiment and analysis, it is clear that all three approaches used for bug prediction are not performing equally on the basis of different comparison criteria and K–S test. It is also found that the second approach, i.e. the potential complexity of code changes/entropy based models (time vs entropy) performance is better than the other two approaches. Based on the results, we found that all three approaches have significant difference; hence we reject the null hypothesis.

7 Managerial applications

The potential complexity of code changes/entropy or potential bugs will help the project manager in calculating the remaining bugs that still exist in the software. It also helps in determining the number of changes required in the software to fix these bugs. Further the changes in the software are also being made due to more usage, more bug fixes, more feature enhancements and addition of new features. These changes will make the software source code complex. The complexity of the code changes will further lead to determine the potential complexity of code changes which will be helpful as a deciding factor for new version/release of the software. These changes further help in determining the software evolution.

This study is quite helpful for the manager who is managing the code. One can continuously draws trends of bug occurrence as well as trends of complexity of code changes. The increase in the complexity may also lead to require more resources as the number of files need to be changed and it is very difficult to remember all of these changes. The predicted bugs/complexity of code changes will be helpful in determining the code maturity. As we are aware that anyone can participate in the development process of open source projects, but it is the duty of the manager or consortia owner to judge the quality of code produced by a particular developer. For example, whenever any contributor of the code wants to fix the bugs, one has to see all changes previously made by all contributors who have contributed towards these changes in source codes. The resource allocation can be optimized by the manager to efficiently produce the quality product by allocating the prospect contributor who is producing less complex codes. This study will help in deciding the release schedule of the software.

8 Threats to validity

In our study, we have considered only those bugs which are affecting the changes in the subsystems under study. We have collected data from the open source repositories available online. A number of changes to the files are affecting the entropy or complexity of code changes. These changes might be lines modified, deleted or added but in our study, we have considered the changes made in the source code as reported in the CVS logs repositories instead of lines deleted/added/modified. The assumptions made during the models building process are violated in the real time scenario. The repair of bugs may introduce further new bugs.

9 Conclusions

Changes in files make the software complex and affect the quality of product but essentially requires for bug repairs, feature enhancement/modification for improvement and new feature introduction in the software. In this paper, we have discussed three approaches namely software reliability growth models, potential complexity of code changes based models and complexity of code changes based models to determine the potential bugs lying in the software. The experiment shows that all the three approaches are sufficient to predict the potential bugs lying dormant in the software. We observed that the potential complexity of code changes based approach is performing well on the basis of goodness of fit criteria. In our experiment, we found that 14, 16 and 11 cases out of total 18 cases each of different approaches respectively show the goodness of fit more than 99 %. We have validated the models using non-parametric K–S test to test the statistical significance of the models proposed in the paper. The K–S test is also confirming the results obtained in the goodness of fit criteria. The K–S test also shows that best cases are lying with time vs entropy or potential complexity of code changes based models with the associated P-Value more than 95 % for two data sets out of three data sets. This study can be further extended to capture all changes for the entire project which will be helpful in predicting the actual number of future bugs for the entire system.

References

Arisholm E, Briand LC (2006) Predicting fault prone components in a java legacy system. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM Press, p 8–17

Chaturvedi KK, Kapur PK, Anand S, Singh VB (2012) Predicting software change complexity using entropy based measures. Paper presented at 6th international conference on quality, reliability, infocomm technology and industrial technology management (ICQRITITM 2012) during 26–28 Nov. 2012 at conference centre, University of Delhi, Delhi

D'Ambros M, Lanza M, Robbes R (2010) An extensive comparison of bug prediction approaches. In MSR'10: Proceedings of the 7th

international working conference on mining software repositories. p 31–41
D'Ambros M, Lanza M, Robbes R (2012) Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Softw Eng 17(4–5):531–577
Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(3):675–689
Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab 28(3):206–211
Graves TL, Karr AF, Marron JS, Siy HP (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661
Gyimothty T, Ferenc R, Siket I (2005) Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng 31(10):897–910
Hassan AE (2009) Predicting faults based on complexity of code change. In: The proceedings of 31st Intl. Conf. On Software Engineering. p 78–88
Hassan AE, Holt RC (2003a) Studying the chaos in code development. In: Proceedings of 10th working conference on reverse engineering
Hassan AE, Holt RC (2003b) The chaos of software development. In: Proceedings of the 6th IEEE international workshop on principles of software evolution
Hassan AE, Holt RC (2005) The top ten lists: dynamic fault prediction. In: Proceedings of ICSM. p 263–272
Huang CY, Kuo SY, Chen JY (1997) Analysis of a software reliability growth model with logistic testing effort function. In: Proceedings of eighth international symposium on software reliability engineering. p 378–388
Kamei Y, Matsumoto S, Monden A, Matsumoto K, Adams B, Hassan A (2010) Revisiting common bug prediction findings using effort-aware models. In: Proc. Int'l Conf. On Softw. Maint. p 1–10
Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware and software reliability. World Scientific Publishing Co., Ltd., Singapore
Kapur PK, Goswami DN, Bardhan A, Singh O (2008) Flexible software reliability growth model with testing effort dependent learning process. Appl Math Model 32:1298–1307
Khoshgoftaar TM, Allen EB, Jones WD, Hudepohl JP (1999) Data mining for predictors of software quality. Int J Softw Eng Knowl Eng 9(5):547–563
Kim S, Zimmermann T, Whitehead J, Zeller A (2007) Predicting faults from cached history. In: Proceedings of ICSE. IEEE, p 489–498
Knab P, Pinzger M, Bernstein A (2006) Predicting defect densities in source code files with decision tree learners. In: Proc. Int'l workshop on mining software repositories. p 119–125
Leszak M, Perry DE, Stoll D (2002) Classification and evaluation of defects in a project retrospective. J Syst Softw 61(3):173–187
Lyu MR (1996) Handbook of software reliability engineering. McGraw-Hill, New York
Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proc. Int'l Conf. On Softw. Eng. p 181–190
Musa JD, Iannino A, Okumoto K (1987) Software reliability, measurement, prediction and application. McGraw-Hill, New York
Nagappan N, Ball T (2005a) Static analysis tools as early indicators of pre-release defect density. In: Proceedings of ICSE. ACM, p 580–586
Nagappan N, Ball T (2005b) Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th international conference on software engineering. p 284–292
Nagappan N, Ball T, Zeller A (2006) Mining metrics to predict component failures. In: Proceedings of ICSE. ACM, p 452–461
Ohba M (1984) Inflection S-shaped software reliability growth model. In: Osaki S, Hotoyama Y (eds) Lecture notes in economics and mathematical systems. Springer, Berlin
Ostrand TJ, Weyuker EJ, Bell RM (2005) Predicting the location and number of faults in large complex systems. IEEE Trans Softw Eng 31(4):340–355
Pham H (2006) System software reliability. Springer series in reliability engineering
Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27(3):379–423 & 623–656
Singh VB, Chaturvedi KK (2012) Entropy based bug prediction using support vector regression. In: Proceedings ISDA 2012–12th international conference on intelligent system design and applications, Nov 27–29, 2012, IEEE Xplore, Kochi. p 746–751
Singh VB, Chaturvedi KK (2013) Improving the quality of software by quantifying the code change metric and predicting the bugs. In: Murgante B et al (eds) ICCSA 2013, Part II, LNCS 7972. Springer, Berlin, pp 408–426
Singh VB, Yadav K, Kapur R, Yadavalli VSS (2007) Considering the fault dependency concept with debugging time lag in software reliability growth modelling using a power function of testing time. Int J Autom Comput 4(4):359–368
The bugZilla project (2013) http://www.bugzilla.org
The Mozilla project (2013) http://www.mozilla.org
Trivedi KS (2001) Probability and statistics with reliability, queuing and computer science applications, 2nd edn. Wiley, New York
Weisberg S (1980) Applied linear regression. Wiley, New York
Xie M (1991) Software reliability modelling. World Scientific Publishing Company Pte. Ltd, Singapore
Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modelling for software error detection. IEEE Trans Reliab 32(5):475–484