Int J Syst Assur Eng Manag (Apr-June 2014) 5(2):155–164
DOI 10.1007/s13198-014-0226-5

ORIGINAL ARTICLE

Predicting the complexity of code changes using entropy based measures

K. K. Chaturvedi · P. K. Kapur · Sameer Anand · V. B. Singh

Received: 10 May 2013 / Published online: 7 February 2014

Abstract Changes in software source codes are inevitable. The source codes of software are frequently changed to meet the user's enormous requirements. These changes are occurring due to bug repairs (BR), enhancement/modification (EM) and the addition of new features (NF). The maintenance task becomes quite difficult if these changes are not properly recorded. The versions of these frequent changes are being maintained using the software configuration management repository. These continuous changes in the software source code make the code complex and negatively affect the quality of the product. In the literature, the complexity of the code changes has been quantified using entropy based measures (Hassan, in: Proceedings of the 31st international conference on software engineering, pp. 78–88, 2009). In this paper, we have proposed a model to predict the potential complexity of code changes using entropy based measures. The predicted potential complexity of code changes helps in determining the remaining code changes yet to be diffused in the software. The proposed model has been validated using seven components of web browser Mozilla. The model has been evaluated using goodness of fit criteria namely R squared, bias, mean squared error, variation, and root mean squared prediction error (RMSPE).The statistical significance of the proposed model has been tested using χ^2 and Kolmogorov–Smirnov (K–S) test. The proposed model is found statistically significant based on the associated p value of the K–S test. Further, we conclude that the rate of complexity diffusion due to BR is found higher in four cases namely Bonsai, Mozbot, tables and XUL. The other components of Mozilla namely AUS, MXR and Tinderbox show increase in complexity due to EM and NF.

Keywords Entropy · Software change complexity · Software configuration management · Software repositories · Open source software

1 Introduction

A software system is continuously subject to maintenance activities, which requires the introduction of new features to satisfy the user needs, fixing faults or adaptation of the system to a new environment. Due to the time pressure, limited effort, and the lack of a disciplined process, these activities tend to deteriorate the software system structure, increasing source code complexity, negatively affecting the system design, and, in general, making the system more difficult to be understood and maintained in the future (Canfora et al. 2010). Studies estimate that maintenance costs are at least 50 % and sometimes more than 90 % of the total cost associated with the system. The relative cost of maintaining software and managing its evolution continuously increasing

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

P. K. Kapur
Amity International Business, Amity University, Noida, UP, India

S. Anand
SS College of Business Studies, University of Delhi, New Delhi, India

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

and represents more than 90 % of the total efforts of the software (Erlikh 2000). The maintenance phase of the software is very crucial for all stakeholders of the software. The requests have been continuously generated from the end user related to feature enhancements, bug fixing/fault repairing and new feature for extending functionalities. The software team assesses the requirement and calculate the effort needed to implement such requests. There is a pressure from the competitors for the early fault fixing and the incorporation of new features. These requirements need a lot of changes which have to be incorporated into the source code during development or maintenance activity to meet the challenges received from the end users. These changes are being recorded in the source code repositories. The number of changes is directly correlated with the errors or faults or bugs in the software. But, it would be difficult to argue that the changes themselves cause errors. Moreover, changes are indicative of the trouble that the programmers face during the code change. Most software evolves over time due to fixing of defects and extending the existing as well as new functionality. Changes in source code are needed to implement the fault fix as well as new feature implementation which makes the software code complex and prone to contain errors.

Developers are distributed across the globe in open source software and they seldom meet each other as opposed to the closed source software. The changes in a distributed scenario make software complex and sometimes causes a failure. These changes can be measured using various complexity measures defined in the literature. Baisli (1980) defines complexity as a measure of the resources expanded by a system while interacting with a piece of software to perform a given task. If the interacting system is a computer, the complexity is defined by execution time and storage requirement need to perform computation. If the interacting system is programmer, the complexity is defined by the difficulty of performing tasks such as coding, debugging, testing and modification. The software complexity is often applied to the interaction between a program and a programmer working on some programming task. Measuring complexity is an important activity which starts from development of code to maintenance (Mockus et al. 1999). Literature is available on source code complexity (McCabe 1976; Halstead 1977), complexity of object oriented design (Chidamber and Kemerer 1994) or functional complexity (Albrecht and Gaffney 1983). These software complexity metrics have been used to predict program length, program development time, number of bugs, the difficulty of understanding a program, and future cost of program maintenance. If the changes are occurring in a specific line or block which does not change the flow or other functionalities of the software, the values of these complexities are not affected. These metrics are not able to capture these changes of the source code. Such changes lead to develop the complexity of code change metric based on changes done in the source code.

Software entropy has been defined by using the second law of thermodynamics which is stated as the second law of thermodynamics. In principle, it states that a closed system's disorder can not be reduced. It can only remain unchanged or increase. A measure of this disorder is called entropy. This law also seems plausible for software systems. As a system is modified, its disorder or entropy always increases (Jacobson et al. 1992). This is called software entropy. The information theory deals with assessing and defining the amount of information in the message (Shannon 1948). The entropy measures the randomness/uncertainty. The theory uses the information in measuring the amount of uncertainty or entropy of the distribution and this has been firstly attempted by Hassan (2009) to quantify the code change process in terms of entropy and predict the bugs based on past defects using entropy based measures. In the present paper, we have proposed a model to predict the potential change complexity in the software. This will be helpful in determining the stage of code maturity. The proposed model also determines the rate at which change complexity diffused in the software due to various types of code changes. The proposed model has been empirically validated using historical code changes data of open source web browser Mozilla's components. The performance of the proposed model has been validated using goodness of fit criteria namely bias, variation, mean squared error (MSE), and root mean squared prediction error (RMSPE).The proposed model has been also statistically validated using Kolmogorov–Smirnov (K–S) and χ^2 test. The rest of the paper is organized in seven sections. Section 2 provides the basics of the code change process and entropy. Section 3 discusses about the proposed model for predicting the potential complexity of code changes. Section 4 deals with goodness of fit criterion. Section 5 discusses about the data collection and processing phase. The results and discussions have been presented in Sect. 6 and finally the paper is concluded with future research direction in Sect. 7.

2 Code change process and entropy

Code change process means to study the patterns of source code modifications. Changes in the source code have been done by developers to implement new features and repair of faults. These patterns have been studied and quantified their degree of complexity over time. Large projects extensively use source code control systems to control and manage their source code (Rochkind 1975). This repository is very rich and continuously updated by the developers

and contains a wealth of information. Data stored in these repositories presents a great opportunity to study the code change process. The data collection costs are minimal since it is collected automatically by the source code control system as the modifications done to the code.

2.1 Types of changes

Various modifications are described as follows:

2.1.1 New feature

This is one of the most important aspects of the software maintenance. New feature requests are coming from the end users/development team and from other competitive teams. The incorporation of changes due to new feature introduction increases the software source code complexity. These changes as well as request have been recorded in software configuration management (SCM) repository.

2.1.2 Feature enhancement

These changes are generally related to formatting of the code, copyright messages and other comments/messages appear in the code as well as during the execution. This will include the enhancement/improvement of the existing features taken as a perfective measure. The feature enhancement will change the existing code and also add some new source code files and hence increase the complexity of code changes.

2.1.3 Bug repairs

Changes in the source code have been done to repair a fault/bug. The changes due to bug repair (BR) may also contribute towards increasing the complexity of code changes as there are many changes needed to fix a fault. Even though, the fault may not get fixed in a single attempt, it also requires interactions among many developers.

The codes are continuously being changed due to any of the above reasons. These changes in the source code are making the software complex. There is need to study the complexity of code changes due to above mentioned reasons.

2.2 Types of code change process

Earlier, Hassan (2009) has identified three variants of the code change process which are as follows:

2.2.1 Basic code change

The changes are recorded based on the number of times the file is modified. These changes are measured at the file level instead of the code level at the fix interval or period. The period can be taken as a day, week, month, year etc. based on the total duration of the project.

2.2.2 Extended code change

Instead of using fixed length period, the basic code change will be extended based on the variable length period. This time period can be divided into three ways i.e., time based periods, modification limits based periods and burst based periods.In time based periods, total length of the project is divided into equal length duration which depends on the clock or calendar time. These partitions can be of any length. In the modification limit based periods, the periods are decided based on the equal number of modifications. The modification limits period depends on the number of modifications in the source code instead of fixed length periods. Usually, the changes do not follow a specific pattern. It generally follows the burst based patterns. Burst based period is depending on the sudden increase or decrease of the modifications in the subsystem. These types of periods have some advantages and disadvantages over one another.

2.2.3 File code change

Files are modified during periods of high change complexity that also have the highest tendency to contain faults. The actual modifications have been considered to determine the period. The modifications may be addition/deletion/modifications in the source code. This can be further extended using extended code change based period instead of a simple calendar/clock based period.

2.3 Complexity of code changes mesasurement

The code change has been studied using information theory based entropy. Entropy is defined as the degree of uncertainty or randomness or complexity of changes in the source code. The information theory deals with assessing and defining the amount of information in the message using the Shannon Entropy (Shannon 1948). Hassan (2009) used this concept of entropy to propose the complexity based measures and predict the bugs which will be coming in the future. The entropy is defined as

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

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

p_k is the probability of change occurrence and defined as the ratio of number of times kth file changed during a period and the total number of changes for all files in that

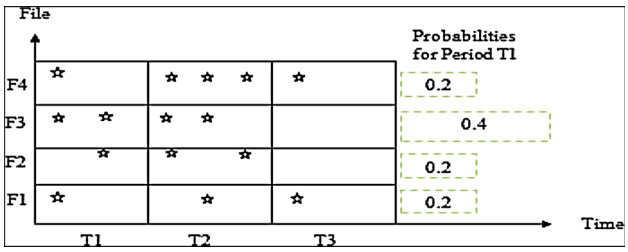

Fig. 1 The pattern of modifications to the file

period. The diagram shown below demonstrates the conversion of change complexity into entropy (Fig. 1).

In the above diagram symbol star represents the change in a file. For a first period, there are five changes occurred in all four files in period T1. The probability of change occurrence in files F1, F2, F3, and F4 will be 1/5 (=0.2), 1/5 (=0.2), 2/5 (=0.4) and 1/5 (=0.2) for first period (Fig. 1). Similarly, probability of change occurrence in files can be calculated for other periods. Hassan (2009) used this entropy and proposed history complexity metric (HCM) based measures for bug prediction based on the pervious faults. It also concluded from the study that the complexity of code changes will get decay in an exponential time period. Further, a benchmark study has been conducted on defect prediction using publicly available datasets and provides an extensive comparison of different bug-prediction approaches (D'Ambros et al. 2010, 2012). This study also proposes some more approaches namely entropy, code churn based approaches and applied existing exponential decay model as well as proposed linear decay and logarithmic decay based models for bug prediction.

3 Proposed model for predicting the complexity of code changes

The modifications in the source code are mainly due to BR, feature enhancements and new feature introduction. The complexity of code changes or uncertainty is diffused in the system due to these changes. This complexity of code changes has been studied and measured using entropy. The aim of this paper is to predict the potential complexity of code changes that the software will have during a long maintenance period.

In this section, we have proposed a model on the line of the Bass model (Bass 1969) to predict the potential complexity of code changes in the software.

We take p is the rate at which entropy/ uncertainty / complexity of code changes is diffused in the code due to new feature introduction (NF)/feature enhancements (EM)and q is the rate at which entropy/uncertainty/ complexity of code changes is diffused due to BRs, the model has been proposed in Eq. (1) with the following assumptions:

(i) Potential entropy /the complexity of code changes $\overline{H}$ is constant.
(ii) The entropy diffusion due to new feature introduction/feature enhancements (EM) is independent and it may introduce bugs.
(iii) At initial time t = 0, means there is no change in file and entropy/the complexity of code changes is zero.

The entropy diffusion or complexity of code changes per unit time can be written as follows:

$$\frac{d(H(t))}{dt} = p(\overline{H} - H(t)) + q\frac{H(t)}{\overline{H}}(\overline{H} - H(t)) \quad (1)$$

where $\overline{H}$ is the potential entropy/potential complexity of code changes to be diffused in software over a period of time and H(t) is the amount of entropy/complexity of code changes at any given time t. Solving above differential equation with initial conditions at t = 0 and H(0) = 0, we get

$$H(t) = \bar{H}\left[\frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p}e^{-(p+q)t}}\right] \quad \text{or} \quad H(t) = \overline{H}\left[\frac{1 - e^{-\phi t}}{1 + \beta e^{-\phi t}}\right]$$

Here $\phi = p + q$ is the entropy diffusion rate due to changes occurred in the source code and $\beta = \frac{q}{p}$ is a constant.

The model parameter has been evaluated using cumulative entropy of the studied subsystem or component using nonlinear regression. This has been tested using statistical measures namely, bias, variation, MSE, RMSPE, R^2 and statistical significance test.

4 Data collection and pre-processing

To empirically validate the proposed model, we have data from the components of subsystems layout and web tools of Mozilla project (The Mozilla project 2012). Mozilla is very popular in the user's community for web browsing. The Mozilla project is using bugZilla (The bugZilla project 2012) for bug reporting. We have selected following subcomponents.

4.1 AUS

Application update service (AUS) is a multi-faceted web based service. This has been used to check the updates of application software available on the server by the client software.

4.2 Bonsai

It is a tool used to perform queries on the contents of a CVS archive. The source code change data have been collected using this tool in the experiment.

4.3 MXR

Mozilla cross reference (MXR) originally uses Linux cross reference (lxr, lxr.linux.no). This tool will help in browsing the repositories of source code namely, CVS server, Mercurial Server, and Subversion Server by cross referencing for the mozilla.org. The source code of the individual files is formatted online and presents the identifiers with hyperlinked interface.

4.4 Mozbot

Mozilla multipurpose extensible modular Perl Bot (Mozbot) provide useful services to the developing communities using other Mozilla web tools, such as displaying changes in the Tinderbox status, displaying information on bug reports from Bugzilla.

4.5 Tinderbox

Tinderbox is a tool used by the Mozilla developer's community to check the compile status of the current source code on various platforms and passes automated test suites. Tinderboxes continually build the latest source code and show the status of these builds. This is helpful in determining the status of the source tree for platform, product, and code branch.

4.6 Tables

Tables are used to design and construct the layout of tables for the user interface.

4.7 XML user interface language (XUL)

An application of XML used to provide the development environment to create user interfaces.

The process flow diagram has shown in Fig. 2 which consist of seven steps starting from the CVS/SVN repository till measuring the performance of the developed model.

The code change data are available in the CVS repository. This historical code change data has been extracted using Bonsai tool of Mozilla from 1998 i.e. the start of the tools made available in the repository. In this study, we have taken a fixed period as 6 months. The number of files

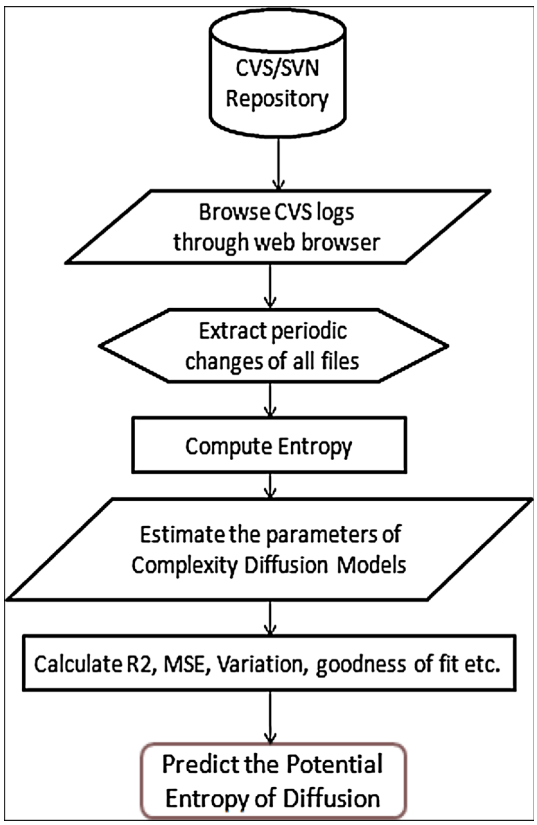

Fig. 2 Process flow diagram for predicting the complexity of code changes or potential entropy

in each of these subsystems is varied in the range of 16–124 files. There are

- 16 files in AUS,
- 124 files in Bonsai,
- 28 files in MXR,
- 57 files in Mozbot,
- 64 files in Tinderbox,
- 29 files in the subsystem "mozilla/layout/tables/", and
- 106 files in the subsystem "mozilla/layout/xul/".

The entropy has been calculated with respect to file change on half yearly basis. This entropy has been used to calculate the potential entropy of the code change using the proposed model.

5 Goodness of fit

The goodness of fit test has been used to provide the statistical significance of the distribution obtained from the predicted values.

Step 1 Null and alternate hypothesis can be stated as below

Table 1 The parameter estimate using the proposed model on various datasets

S. no.	Dataset	$\overline{H}$ (predicted)	ϕ	β	p	q	Maturity period of (unit)	Entropy at maturity level
1	AUS	9.309	0.094	0.288	0.073	0.021	48	8.94
2	Bonsai	64.279	0.030	3.057	0.007	0.023	106	60.59
3	Mozbot	12.327	0.198	9.329	0.019	0.179	29	11.93
4	MXR	9.298	0.114	0.001	0.1139	0.0001	28	8.90
5	Tinderbox	74.791	0.020	0.779	0.011	0.009	70	47.26
6	Tables	60.105	0.050	1.731	0.018	0.032	59	52.23
7	XUL	53.970	0.094	3.057	0.023	0.071	49	51.85

1 Unit = 6 months

The null hypothesis (H_0): The observed and estimated values of the complexity of code changes are equal.

The alternate hypothesis (H_1): The observed and estimated values of the complexity of code changes are not equal.

Step 2 Calculate the value of χ^2.

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}$$

where o_i and e_i are the observed and estimated value of complexity of code changes at (k − 1) degree of freedom.

Table 2 Goodness of fit measures of various datasets

S. no.	Dataset	R^2	Bias	MSE	Variation	RMSPE
1	AUS	0.997	−0.006	0.017	0.102	0.102
2	Bonsai	0.997	0.078	0.151	0.389	0.397
3	Mozbot	0.979	−0.053	0.303	0.560	0.563
4	MXR	0.959	0.028	0.179	0.434	0.435
5	Tinderbox	0.971	0.195	1.003	1.001	1.020
6	Tables	1.000	0.029	0.011	0.103	0.107
7	XUL	1.000	−0.014	0.009	0.098	0.099

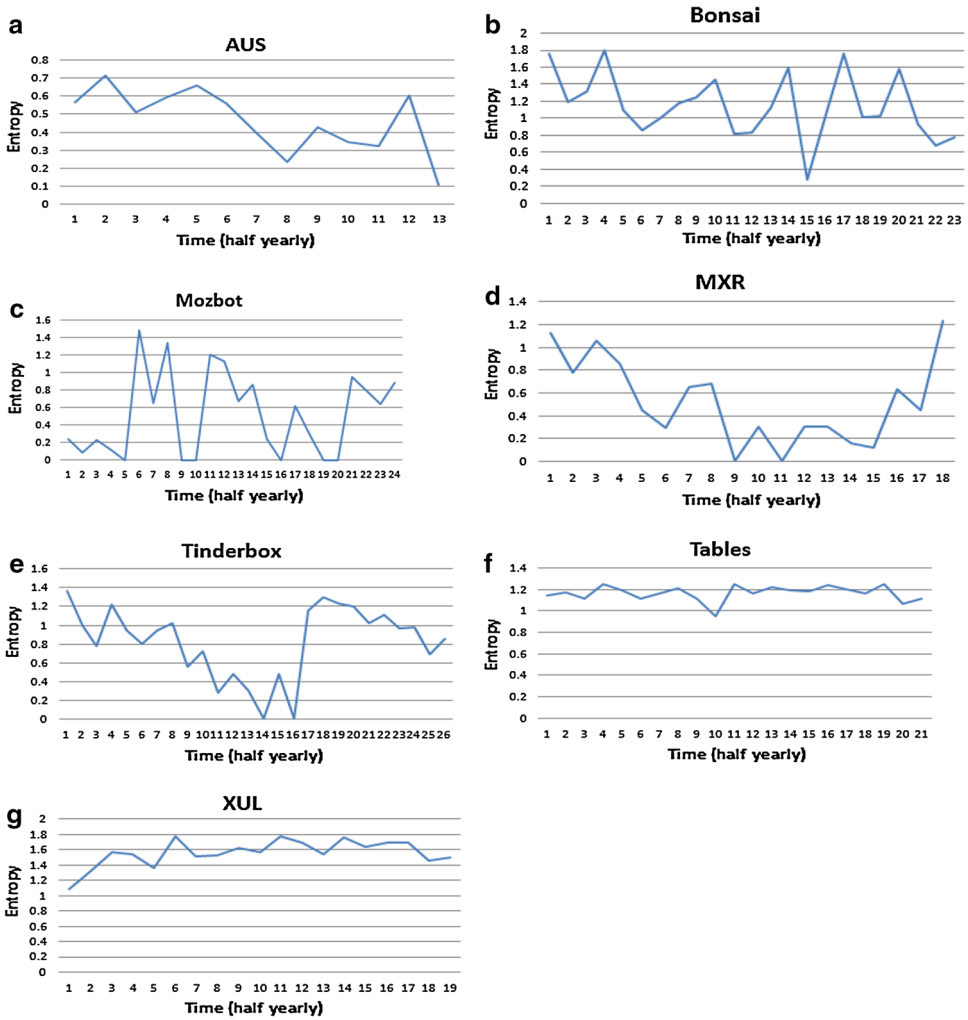

Fig. 3 **a** Complexity of code changes for AUS. **b** Complexity of code changes for Bonsai. **c** Complexity of code changes for Mozbot. **d** Complexity of code changes for MXR. **e** Complexity of code changes for Tinderbox. **f** Complexity of code changes for tables. **g** Complexity of code changes for XUL

Step 3 Compare the estimated value with the tabulated value of the k − 1 degree of freedom. If the calculated value is less than the estimated value, the alternate hypothesis is rejected and the null hypothesis is accepted.

5.1 Kolmogorov–Smirnov (K–S) Test

This is a non-parametric test and able to provide the distance between the observed and predicted curve with the certain confidence level in terms of probability.

6 Results and discussions

The parameters of the diffusion model have been estimated using SPSS software and the results are shown in Table 1. Table 1 shows the value of the potential complexity of code changes/entropy, stabilized entropy, stabilization period, rate of entropy diffusion due to feature enhancement/new feature addition and BR. The value of p is the rate at which entropy/uncertainty/complexity of code changes is diffused in the code due to new feature introduction (NF)/feature enhancements (EM) and q is the rate at which entropy/uncertainty/complexity of code changes is diffused due to BR. It is clear from the table that for some data sets p is more than q and for some q is more than p.

The value of p is larger than the value of q in AUS, MXR and Tinderbox which means in these components the complexity of code changes have been introduced mainly due to feature enhancement and new feature introduction. The AUS component is mainly used by the client software to check the updates of the application software. This component is being made rich by providing new functionalities. It means that the changes in the source code are mainly due to the inclusion of new feature or feature

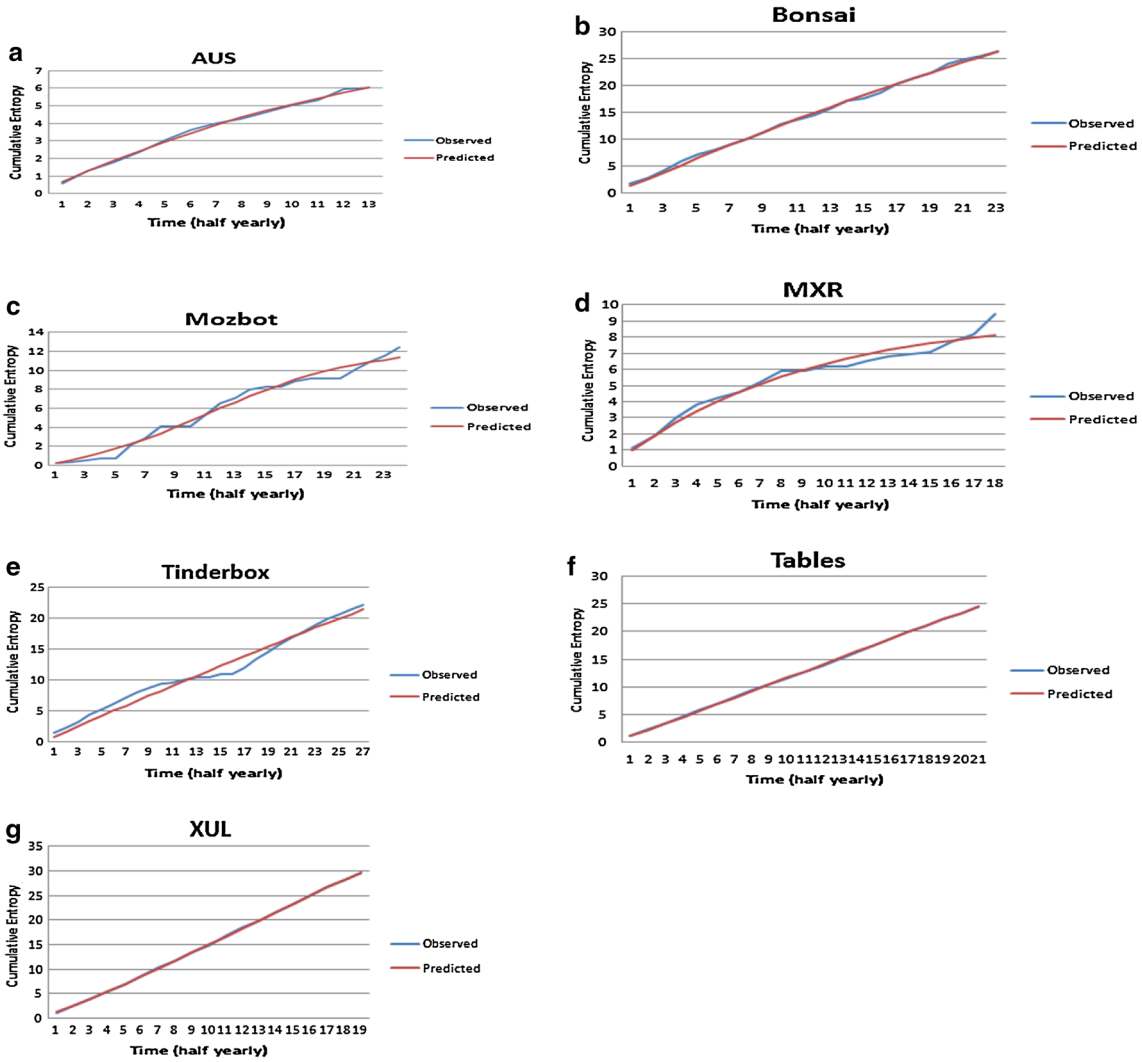

Fig. 4 **a** Goodness of fit curve for AUS. **b** Goodness of fit curve for Bonsai. **c** Goodness of fit curve for Mozbot. **d** Goodness of fit curve for MXR. **e** Goodness of fit curve for Tinderbox. **f** Goodness of fit curve for tables. **g** Goodness of fit curve for XUL

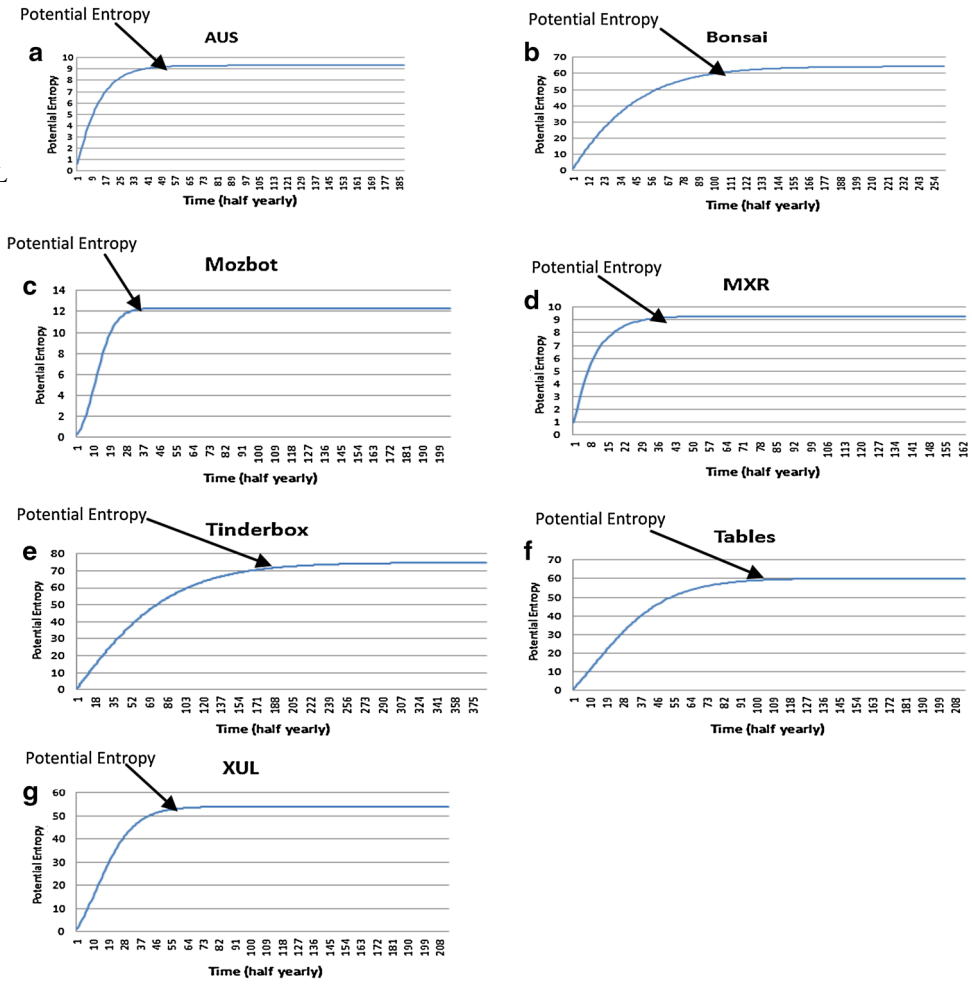

Fig. 5 **a** Potential complexity for AUS. **b** Potential complexity for Bonsai. **c** Potential complexity for Mozbot. **d** Potential complexity for MXR. **e** Potential complexity for Tinderbox. **f** Potential complexity for tables. **g** Potential complexity for XUL

enhancements. The MXR originally uses the source code of lxr. The bugs are reported in these components are being originally fixed in the lxr and only changes in the MXR components are being made to improve the existing features and addition of new features. The new feature introductions include as an example to provide interactive web based GUI front end for browsing the source code using cross referencing. The Tinderbox is used to check the compile status of the current source code on various platforms status of the build source code various computers. This component provides the status details of various builds such as a nightly build, dependent build, leak text build, compile and run tests, static analysis. Most of these features are only update/format oriented rather bug fix oriented. Our experimental results validate the practical occurrences in the components.

On the other side, the value of p is smaller than the value of q in Bonsai, Mozbot, tables and XUL which means in these components the complexity of code changes have been introduced mainly due to bug fixes/repairs. These components are mainly developed by Mozilla. The releases are mostly due to bug fixes.

Various goodness of fit criteria results has been shown in Table 2, which indicates that the proposed model fits the observed data well. Table 2 shows the values of various statistical parameters namely bias, MSE, variation, RMSPE and R^2. The value of R^2 is more than 0.95 for all the components. For the components tables and XUL the value is 1. The other statistical parameters show that the model is able to capture the entropy based complexity of code changes and provide the better goodness of fit.

The pattern of entropy as complexity of code changes has been shown in the Fig. 3a–g for various components of Mozilla. In most of the components, the behaviour of entropy is non-uniform except in layout components i.e., XUL and tables. In these components, minor variation in the entropy has been observed for two consecutive periods. The goodness of fit curve between observed and predicted entropy has been plotted for these components in Fig. 4a–g. These figures signify the better goodness of fit of the model

Table 3 Goodness of fit for complexity of code changes prediction

Components	χ^2 (calculated)	χ^2 (tabulated)	Degree of freedom
AUS	0.0431	21.026	12
Bonsai	0.4762	33.924	22
Mozbot	1.9269	35.172	23
MXR	0.4903	27.587	17
Tinderbox	3.7407	38.885	26
Tables	0.0447	31.410	20
XUL	0.0573	28.869	18

Table 4 Tests for statistical significance using Kolmogorov–Smirnov (K–S) two sample test

Components	D value	Associated p value
AUS	−0.0769	1.0000
Bonsai	−0.0435	1.0000
Mozbot	−0.1250	0.9868
MXR	−0.1667	0.9448
Tinderbox	−0.1111	0.9936
Tables	−0.0476	1.0000
XUL	0.0526	1.0000

as the most of the points in these charts are overlapping with each other in respect of observed and predicted values.

Figure 5a–g shows the potential complexity of code change in the software for the components under study. We observed from the graph that the value of complexity of code changes/entropy increases with respect to time due to feature enhancement/new feature addition and BR but it got stabilizes/matures at some point of time and after that the value increases very slowly to meet the predicted potential entropy. This trend follows for all the data sets. We have also calculated the value of the complexity of code changes/entropy which stabilizes/matures the development of the product.

To test the statistical significance of the proposed model, we have applied non-parametric χ^2 and K–S test and the results of these statistical tests are shown in Tables 3 and 4 respectively.

In Table 3, the second, third and fourth columns contain the calculated value of χ^2, tabulated value of χ^2 at the 5 % level of significance and degree of freedom respectively.

In all the datasets, the calculated value of the χ^2 statistic is less than the tabulated value of the χ^2 statistic with respective degree of freedom at the 5 % level of significance. In this way, we can say that the null hypothesis is accepted. Thus, we can conclude that the predicted potential complexity of code changes/entropy has been found statistically significant.

The K–S test shows the distance as D value with the confidence provided by associated p value. In all the datasets, the probabilities are approximately one which shows that the proposed model is able to fit the data very well and the predicted value of potential entropy or complexity of code changes are found to be statistically significant.

7 Conclusion

In this paper, the first one and a novel approach has been proposed to predict the potential complexity of code changes in the software. The diffusion of change complexity has been validated using seven components namely AUS, Bonsai, Mozbot, MXR, Tinderbox, tables and XUL of Mozilla project. Experimental results on the basis of different performance measures show that the diffusion of the complexity of code changes in software can be predicted significantly using the proposed model. The entropy based diffusion of change complexity is a good predictor of potential entropy or potential change complexity i.e., the remaining code changes yet to be diffused in the software. The value of R^2 is over 95 % of all cases. For statistical validation, the proposed model is found statistically significant based on χ^2 and K–S test for all the datasets. Our study shows that the proposed measure of software change complexity based on entropy can be applied in predicting the remaining requirements or code changes yet to be diffused in the software and hence it will help in managing the software product. In the future, we will extend this work on more data sets and at finer grain levels where we can consider code level changes.

References

Albrecht AJ, Gaffney JR (1983) Software function, source lines of code and development effort prediction: a software science validation. IEEE Trans Soft Eng 9(6):639–648

Baisli VR (1980) Qualitative software complexity models: a summary. In: Tutorial on models and methods for software management and engineering. IEEE Computer Society Press, Los Alamitos

Bass F (1969) A new product growth model for consumer durables. Manag Sci 15(5):215–227

Canfora G, Cerulo L, Di Penta M, Pacilio F (2010) An exploratory study of factors influencing change entropy. In: The 18th IEEE international conference on program comprehension, ICPC 2010, Braga, Minho, Portugal, 30 June–2 July 2010. IEEE Computer Society, Washington, DC, pp. 134–143

Chidamber AJ, Kemerer CF (1994) A metric suite for object oriented design. IEEE Trans Soft Eng 20(6):476–493

D'Ambros M, Lanza M, Robbes R (2010). An extensive comparison of bug prediction approaches. In: Proceedings of the seventh IEEE working conference on mining software repositories (MSR07). IEEE CS Press, pp. 31–41

D'Ambros M, Lanza M, Robbes R (2012) Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empir Softw Eng 17(4–5):531–577

Erlikh L (2000) Leveraging legacy system dollars for E-business. (IEEE) IT Pro, May/June 2000, pp 17–23

Halstead MH (1977) Elements of software science. Elsevier North Holland, New York

Hassan AE (2009) Predicting faults based on complexity of code change. In: proceedings of the 31st international conference on software engineering, pp 78–88

Jacobson I, Christerson M, Jonsson P, Overgaard G (1992) Object oriented software engineering: a use case driven approach. ACM Press, Addison Wesley, New York, Reading, pp 69–70

Kagdi H (2007) Improving change prediction with fine-grained source code mining. In: Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering, Atlanta, Georgia, USA, 05–09 November 2007

Kagdi H, Maletic JI (2006) Software-change prediction: estimated+actual. In: SOFTWARE-EVOLVABILITY '06 proceedings of the second international IEEE workshop on software evolvability. IEEE Computer Society, Washington, DC, pp 38–43

Kagdi H, Maletic JI (2007) Combining single-version and evolutionary dependencies for software-change prediction. In: Proceedings of the fourth international workshop on mining software repositories held during 20–26 May 2007

McCabe TH (1976) A complexity measure. IEEE Trans Soft Eng 2(6):308–320

Mockus A, Eick SG, Graves TL, Karr AF (1999) On measurement and analysis of software changes. Technical report, National Institute of Statistical Sciences

Rochkind MJ (1975) The source code control system. IEEE Trans Soft Eng 1(4):364–370

Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27(379–423):623–656

The bugZilla project (2012). http://www.bugzilla.org. Accessed 30 Nov 2012

The Mozilla project (2012). http://www.mozilla.org. Accessed 30 Nov 2012