Jayadeep et al., International Journal of Advanced Research in Computer Science and Software Engineering 4(1), January - 2014, pp. 1-6

Volume 4, Issue 1, January 2014 **ISSN: 2277 128X**

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Prediction of Temporal Bug Patterns of Debian Using Markov Model

Jayadeep Pati*, K K Shukla

Department of Computer Science & Engg.

IIT(BHU), Varanasi, India

Abstract: Predicting the temporal bug patterns is a challenging aspect in software engineering. This paper focuses on modelling the growth in bug numbers as Markov Chain with varying bug growth patterns representing different states. Real software bug data which shows random behaviour can be considered to be generated by an underlying non-stationary random process. The paper demonstrates the distribution of temporal bug patterns of an open source software and also effectiveness of Markov Models in predicting bug growth patterns. As a generalised model this can also be applicable for bug growth patterns for closed source software. The model will be useful for software project managers for taking timely decisions such as effort investment and allocation of resources. Another use of this model will be reduction of testing effort. The end users by knowing the possible bug patterns in the system in advance can take timely precautions to cope with the loss due to system failure. The performance analysis is done on bug data of Debian Operating System extracted from Ultimate Debian Database (UDD) which is publicly available.

Keywords— Debian, Bug, Temporal Pattern, Markov Model, Transition Matrix.

1 Introduction

The bugs in the software from the software engineering point of view may be a design bug, testing bug, semantic bug, application bug, syntactic bug, security bug, memory bug [12], etc. The bugs are responsible for 40% of software failures [2]. It is very difficult to predict the bug growth patterns as it sometimes show random behaviour. The bug in software can be thought to be generated by a stochastic process. There also previous studies on software defect prediction based on static code attribute [1, 2, 3, 4, 5, 6, and 7]. In these papers software metrics are used to predict the software fault content and software fault proneness. There are also studies on prediction of software defect based on bug numbers [8, 9]. The previous studies on bug number predictions are based on traditional approaches like ARIMA [2] which is applied to predict a stationary time series data. In another paper [10] Polynomial regression technique is applied to model the cumulative growth of eclipse bug numbers. In this paper we have a different approach as we are predicting bug patterns rather than bug values which are of more practical significance. Markov models are frequently used for stochastic modelling [13]. The Markov models are also popular in predicting other time series applications [14, 15]. This paper we have used Markov model to predict the bug growth patterns in Debian, an open source operating system[17]. We have associated different states to the increasing and decreasing bug patterns to model them as Markov chain. The entire bug number data is clustered using K Median clustering to generate the K clusters which represents different states of Markov Chain. The rest of the paper is organized as follows: Section 2 presents a description of Markov Chain Modelling. Section 3 describes about Data Collection.Section 4 describes about Fromation of Markov Model Generation. Section 5 describes on Performance Evaluation. Section 6 describes about Results and Interpretation. Section 7 describes about Application of Model Section 8 concludes the paper and shows direction for future work.

2 Markov Chain Model

Markov chain is a stochastic process which can be parameterized by empirical estimation of transition probabilities between discrete states in the system under study[14]. In the 1st order Markov chain the next state only depends on the current state. On higher order Markov chains the next state depends on two or more preceding states.

Let Z(t) be a stochastic process, Containing discrete States Space S= 1 ,2, 3,…..k, where time series is t1 < t2 <.... tn. The conditional probability can be represented as

$$P(x_{t+1}=i_{t+1}|x_0=i_0,x_1=i_1,\ldots\ldots x_t=i_t)=P(x_{t+1}=i_{t+1}|x_t=i_t)$$

This is called one step transition probabilities of markov chain. These probabilities can be written as

$P_{ij}=P(x_{t+1}=i|x_t=j)$ for i and j in S.

The matrix$P=(P_{ij})_{k\times k}$ is called transition matrix. The elements of P must satisfy two properties:

$$0<p_{ij}<1\forall i,j\in S \qquad \sum_{i=1}^{k} p_{ij}=1,\forall j\in S$$

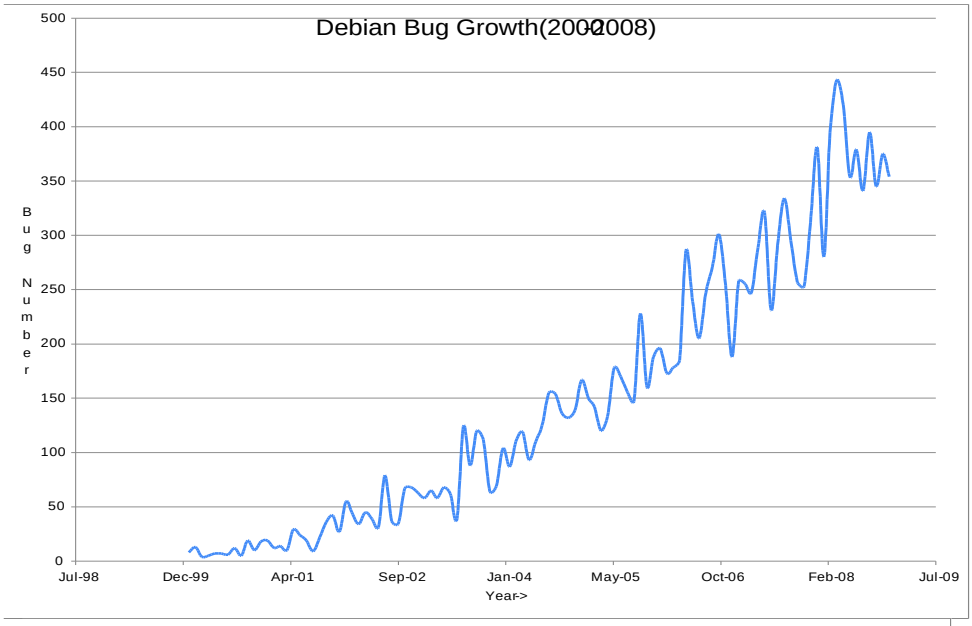
The 1st order markov chain model can be represented as $X_{t+1}=PX_t$. For 3 states, the first order transition matrix P has a size of 3×3. The transition matrix for 2nd order P has a size of 9×3. The 3rd order transition matrix has a form 27×3.

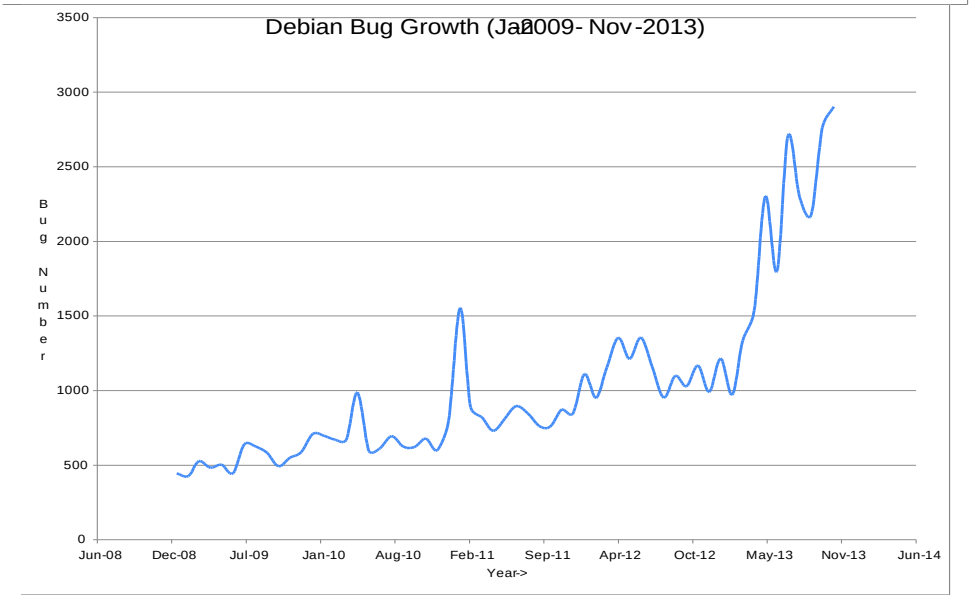
2.1 Conversion from Higher Order to 1st Order

Higher order markov chain depend on historical information. For example a 3rd order markov chain depend on last 3 states visited. At a given time t, the probability of state s at time t+1 is $P(s|x,y,z)$, where x,y,z are current and two preceding states. Our focus is on 1st -order Markov chain because any higher-order Markov chain can be converted into a 1st order markov chain[13]. For instance if in a 3rd order markov chain, the states visited are ...a,b,c,d,a,b,e,a,...and if we are interested in predicting probability of next state to d, it depends on its recent history: b,c,d. To construct the equivalent 1st order markov chain, we have to name the states in such a way that history can be reconstructed from the state name. We have to collect three states worth of history and encode the information into name of the current state. When the third-order Markov chain is in state d, the equivalent 1st order one would be in state acd and the states visited will be...? ? a, ? ac, acd, cda, dab, abe, bea, ea? , a? ? .... Each of the state names gives us the same information that the third-order Markov chain possess at the same moment. So, any information that we will get in higher-order representation can be deduced from the longer name in the 1st order representation.

Data Collection

In this paper, we have studied the bug number growth per month, and we have obtained the original data from the public Debian bug data that is available Ultimate Debian Database[4]. We have analyzed the Debian bug data from January 2000 to February 2013. The original data in the bug tracking database is like this: "id";"Last Modified": "607021";" 2010-12-14 ". There are different types of bugs available found in UDD like PHP Bugs, Ruby Bugs, Localisation bugs ,etc. We have considered all types of bugs in our analysis. In order to do time series analysis, we transform the data into monthly bug count. Here, we have used 166 monthly data. The bug series has significantly increasing trend from 2000-2013. The bug growth behaviour from 2009- 2013, 2000 - 2008 and 2000-2013 ig given in figure 1.





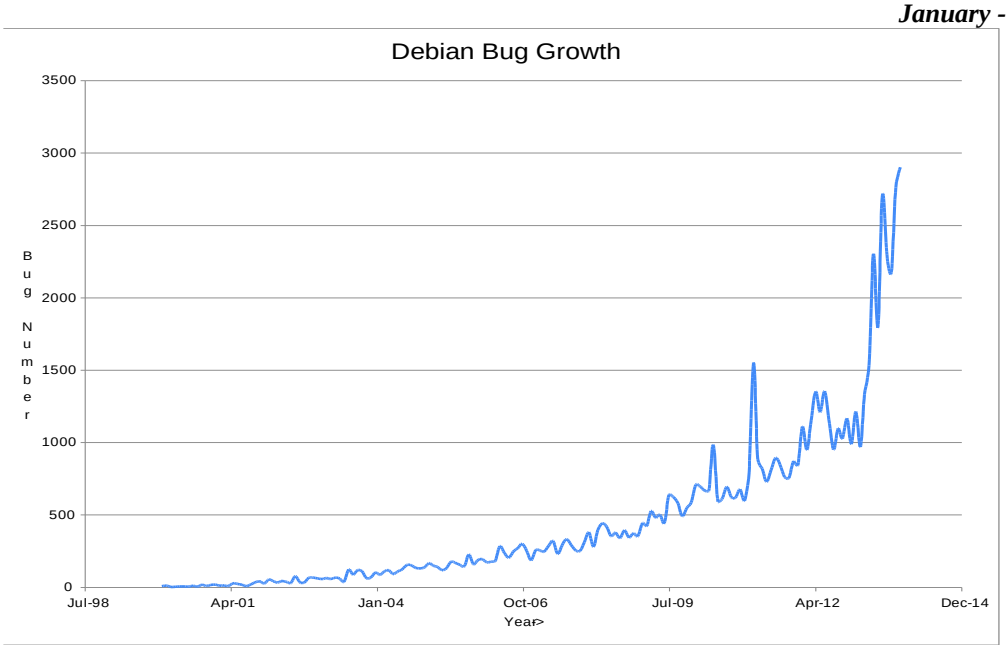


Figure 1: Time Series behaviour of Debian bugs

Stationarity Test

Here, ADF (Augmented Dickey-Fuller Test) is used to test the behavior of Debian bug number series. The ADF test is used to check the presence of a unit root which leads to violation of assumption of classical linear regression in autoregressive time series models. Presence of a unit root indicates the non stationary behavior of a time series. The standard Dickey Fuller test estimates the equation as given below:

$\Delta y_t = y_{t-1} + \varepsilon_t$

The Dickey Fuller test is only valid for AR(1) processes. If the time series is correlated at higher lags, the augmented Dickey Fuller test performs a parameter correction for higher order correlation, by adding lag differences of the time series. In this paper The ADF test is performed in the [web:reg] [16] (an Add-In to Excel written by Kurt Annen.). The unit-root existence is checked in the following formulations:

1. With Constant or intercept
   The Equation is: $\Delta y_t = \alpha + \gamma y_{t-1} + \varepsilon_t$
2. With constant+trend ()
   The Equation is: $\Delta y_t = \alpha + \gamma y_{t-1} + \beta \times t + \varepsilon_t$

The ADF test is basically $H_0 : \gamma = 0$.

If the test statistic is less (Due to non-symmetrical nature of the test absolute value is not considered) than (a larger negative) the critical value, then the null hypothesis of $H_0 : \gamma = 0$ is rejected and it indicates absence of unit roots. After statistical calculations the result is given in figure 3.

Augmented Dickey-Fuller Unit Root Test on tseries(2000-2013)

Null Hypothesis: tseries has a unit root
Exogenous: Constand and linear Trend
Lag Length: 8 (Automatic Based on AIC, MAXLAG=10)

	t-Statistic	Prob.*
Augmented Dickey-Fuller test statistic	2.443167	1.000000
Test critical values: 1% level	-4.017232	
5% level	-3.438553	
10% level	-3.143581	

*MacKinnon (1996) one-sided p-values.

Augmented Dickey-Fuller Test Equation
Dependent Variable: D(tseries)
Method: Least Squares
Date: 24-Dec-13 Time: 2:50:21 PM
Included observations: 158 after adjusting endpoints

Figure : Tabulated Results of Stationarity Test

From the table the non-stationary behavior of the Debian time series is confirmed.

4 Generation of Markov Model

To model the bug growth pattern as Markov model we need different states and transition probability between the states.

Formation of Markov States

Here we have considered the frequent increasing and decreasing "behavior of" bug growth patterns as different states. The entire bug data series contain random number of increasing and decreasing patterns. The change in bug number is calculated by taking first difference in the bug number time series. Taking the 1st foroward difference makes the series stationary as shown by ADF test similar to section 3.1. Therefore Markov chain model is justified. The increasing and decreasing bug pattern from 2000-2013 is given in figure 3. The sorted order of change in bug number series is also plotted in figure 4, which shows symmetrical distribution of increasing and decreasing bug patterns. We have used K-Median clustering to cluster the entire increasing and decreasing patterns into K clusters due to the symmetric behavior of data series. Each cluster represents one state of Markov chain. Here, we have taken K=5, hence we get 5 clusters. Each of them represents one state of Markov chain. After applying clustering to the changing bug growth pattern we get 5 states:

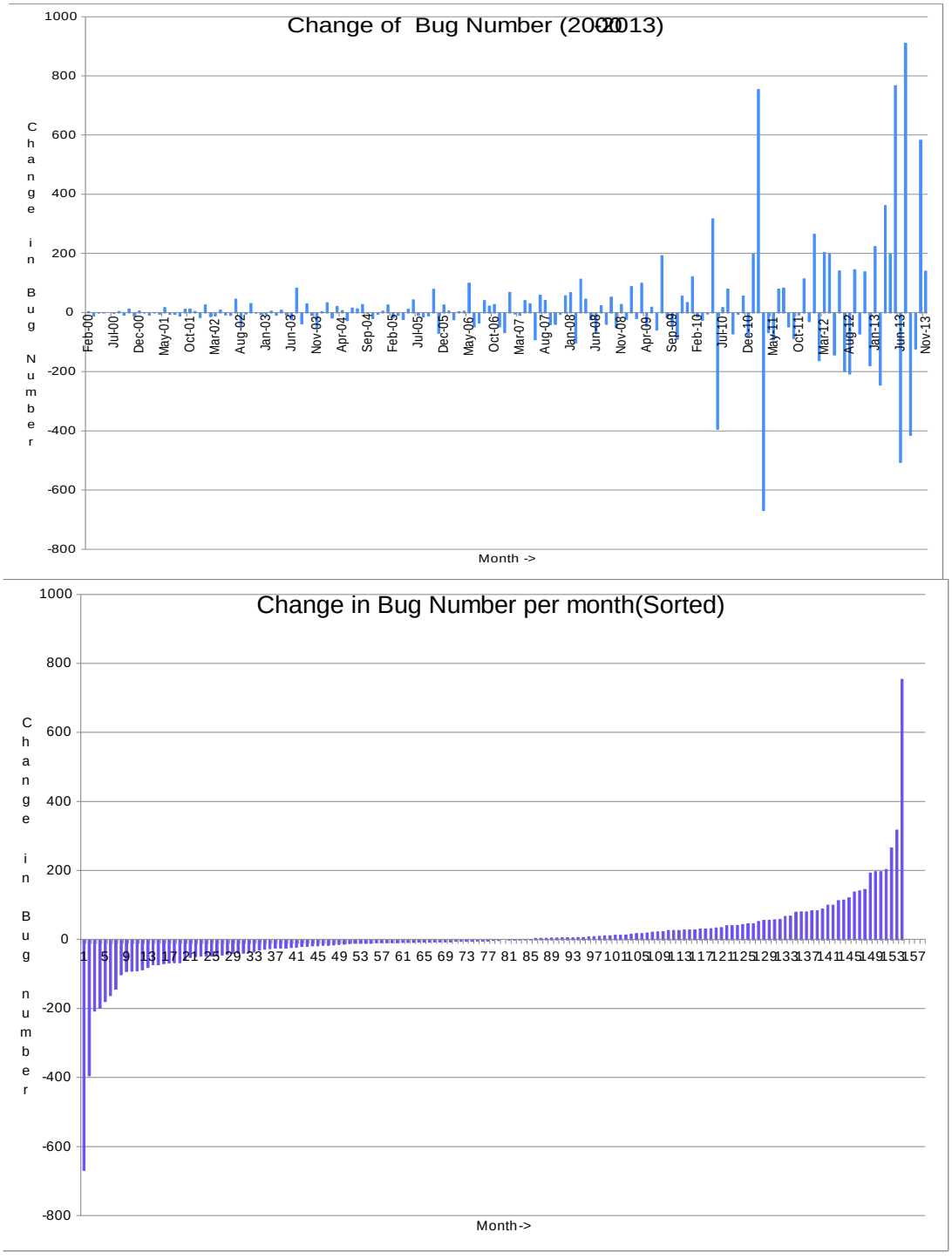


Figure : Plot for Change in Bug Number per Month

Figure : Plot for Change in Bug Number(Sorted)per Month

1. HD (High Decrease)
2. LD (Low Decrease)
3. I (Insignificant change)
4. SI (Simple Increase)
5. HI (High Increase)

The upper threshold and lower threshold values of different states as decided by K-median clustering is given in Table 1.

Table 1: Table Showing Threshold Values of Different States

States	Lower Threshold	Upper Threshold
HD	-667	-38
LD	-36	-1
I	0	15
SI	17	70
HI	81	756

After getting the state information now, the Markov model representation of Debian bug number series is given in figure 5. The bidirectional arrows representing the transition probabilities.

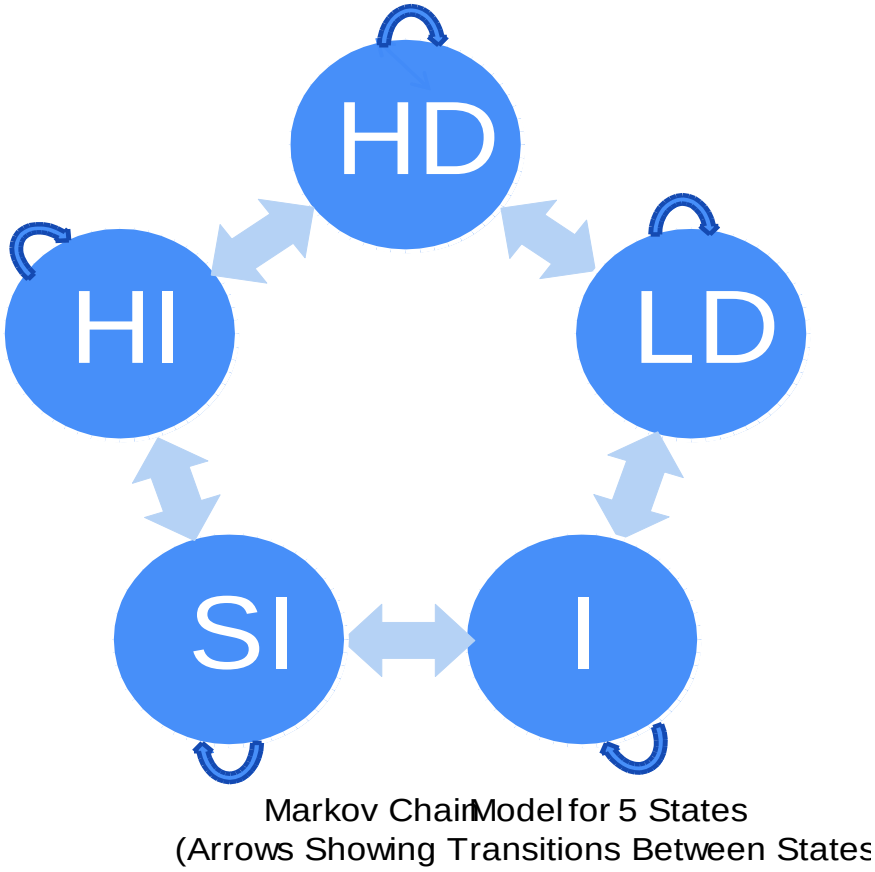


Figure 5: Markov Model Representation of Debian Bug Number Series

Use of clustering for determining number of states is further elaborated in the next section.

K- Median clustering

In data mining, K- median clustering is a variation of K-means clustering where instead of calculating cluster mean to determine its centroid, the median is chosen [11]. The goal of K-median clustering is to create distinct groups based on differences in the data. After clustering, we get K-distinct groups with different characteristics.

We have performed K-Median clustering using MultiExperiment Viewer (MeV) of the TM4 suite of tools [10]. The TM4 software system represents a comprehensive, extensible, open-source, and freely available collection of tools which is used in a wide range of laboratories conducting microarray experiments [10]. MultiExperiment Viewer is a Java application designed to allow the analysis of microarray data to identify patterns of gene expression and differentially expressed genes. MultiExperiment Viewer supports many clustering techniques like HCL - Hierarchical clustering, K-means Clustering, K- Median clustering and other data mining techniques. We have used K- median clustering considering Euclidian distance between data items as our distance metrics. The reason for choosing K-median is that it is not sensitive to outliers and an exploratory Analysis of the Debian data showed existence of significant outliers. For other data sets one a choose an appropriate clustering algorithm for identifying the number of states.

Distribution of Markov States

The next step is to analyse the distribution of different states in a particular time span. We have divided the entire series into two parts. The first part from 2000-2008, the second part from 2009-2013. Now the distribution of different states in these durations is given in figure 6 and figure 7.

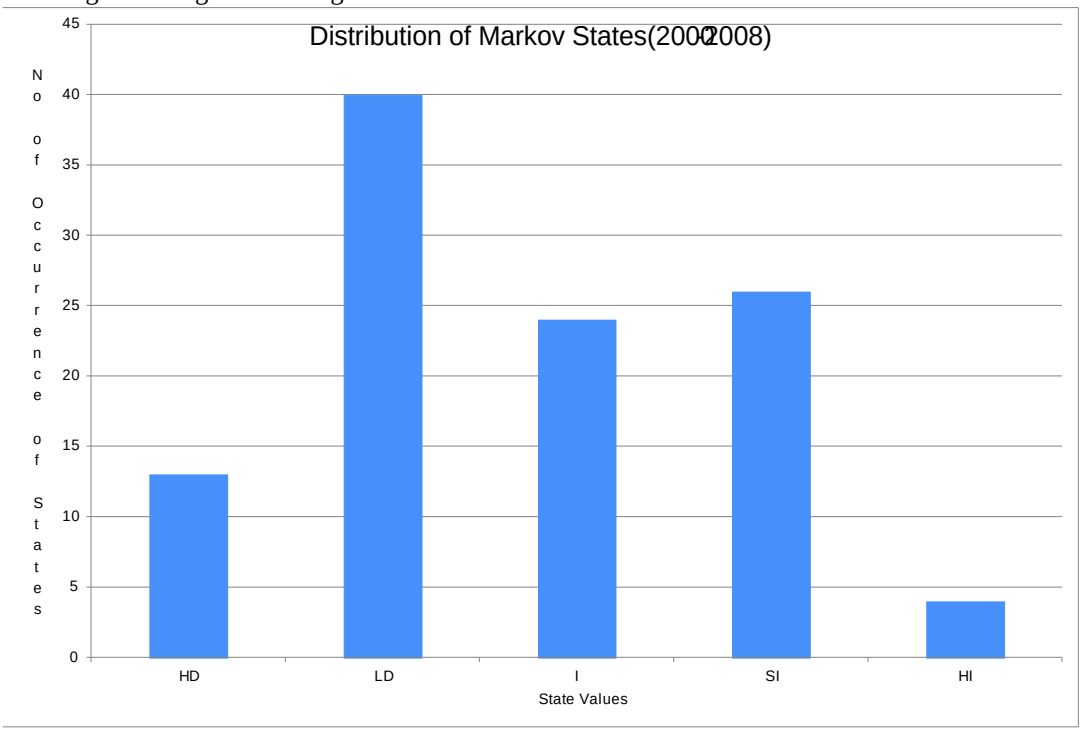


Figure : Distribution of Markov States (2000-2008)

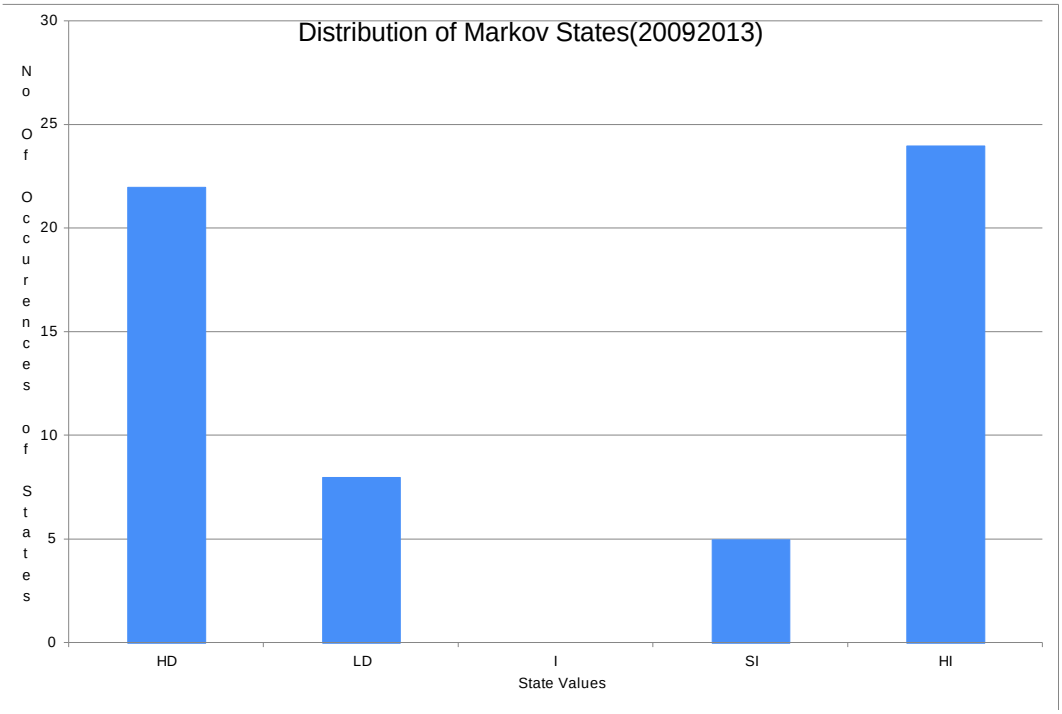


Figure : Distribution of Markov States(2009-2013)

From figure 6 we found that from 2000-2008, the most frequent state is LD followed by SI and I and least frequent states are HI and HD. But as shown in figure 7, from 2009-2013 the most frequent states are HI followed by HD which shows large values of increasing and decreasing bug patterns. An open source software is initially lunched with small number of people working with fewer components and less number of users using the application.This leads to small and insignificant changes in bug numbers. But gradually the number of contributors, users increases which intern increases the complexity of software and also the number of added components[2]. The numbers of change requests and commits also increases which leads to large changes in bug numbers.

Formation of Transition Matrices

After getting the corresponding state values for bug number data, the next step is to design them as markov chain. We have arranged the data as first order and second order markov chain. In first order Markov chain the next state is only dependent on current state and no history is required. The second order Markov chain requires history for predicting the next state. Sample data designed for modeling them as markov chain is given in figure 8.

		Markov State Representation		
	Training 1st Order		Training 2nd Order	
	Current State	Next State	Current State	Next State
1	SI	HD	SI HD	HD HI
2	HD	HI	HD HI	HI SI
3	HI	SI	HI SI	SI LD
4	SI	LD	SI LD	LD HD
5	LD	HD	LD HD	HD SI
6	HD	SI	HD SI	SI HD
7	SI	HD	SI HD	HD SI
8	HD	SI	HD SI	SI HD
9	SI	HD	SI HD	HD SI
10	HD	SI	HD SI	SI LD
11	SI	LD	SI LD	LD HI
12	LD	HI	LD HI	HI LD
13	HI	LD	HI LD	LD HI
14	LD	HI	LD HI	HI HD
15	HI	HD	HI HD	HD SI

Figure 8: Markov State Representation

Our next step is to calculate the transition probabilities for all types of markov chains. We estimate the transition probability matrix P_i by the following method. Given the data series, the transition frequency is counted from the states in the sequence at time t = m -i+ 1 to the states in the jth sequence at time t = m + 1 for $1 \leq i \leq n$. Thus we construct the transition frequency matrix for the data sequences. After making normalization, the estimates of the transition probability matrices P can also be obtained[2].

For 2nd order Markov model the data sets are first converted into 1st order Markov model by the technique given in previous section and then the transition frequencies are calculated.

In our case as we have 5 states we have transition probability matrix 5×5 for first order Markov chain model has been shown in Figure 9. The second order transition probability matrix is of size 25×5, which is shown in Figure 10.

	TRANSITION PROBABILITY(1ST ORDER)	HD	LD	I	SI	HI
1	HD	0.266667	0.133333	0	0.066667	0.4
2	LD	0	0.4	0	0.2	0.4
3	I	X	X	X	X	X
4	SI	0.5	0	0	0	0.5
5	HI	0.692308	0.076923	0	0	0.230769

Figure 9: Transition Probability for 1^{st} orderMarkov Model (Train Period: 2012 - 2012)[X: Lack of Transition]

	TRANSITION PROBABILITY (2^{nd} ORDER)	HD	LD	I	SI	HI
1	HD HD	0.25	0.25	0	0	0.5
2	HD LD	0	0	0	0.5	0.5
3	HD I	X	X	X	X	X
4	HD SI	0	0	0	0	1
5	HD HI	0.5	0	0	0	0.5
6	LD HD	X	X	X	X	X
7	LD LD	0	0.5	0	0	0.5
8	LD I	X	X	X	X	X
9	LD SI	1	0	0	0	0
10	LD HI	0.666667	0.333333	0	0	0
11	I HD	X	X	X	X	X
12	HI LD	X	X	X	X	X
13	I I	X	X	X	X	X
14	I SI	X	X	X	X	X
15	I HI	X	X	X	X	X
16	SI HD	0	0	0	0	1
17	SI LD	X	X	X	X	X
18	SI I	X	X	X	X	X
19	SI SI	X	X	X	X	X
20	SI HI	1	0	0	0	0
21	HI HD	0.375	0.125	0	0.125	0.375
22	HI LD	0	0.5	0	0	0.5
23	HI I	X	X	X	X	X
24	HI SI	X	X	X	X	X
25	HI HI	1	0	0	0	0

Figure : Transition Probability for 2^{nd} orderMarkov Model (Train Period: 2010 - 2012)[X: Lack of Transition]

From the transition probability matrix we can get the most probable next state values from a given a state values. The state which has highest transition probability P_{Max} {State1 → State2} in a row is most likely to occur as next state. Figure 9,10 shows the transition probability matrix for the markov model trained for 3 years(36 months) from 2010 - 2012. for 1^{st} order and 2^{nd} order Markov chain respectively. Below figure 11,12 shows the most probable state from a given state for a Markov model trained for the above period.

CURRENT STATE	MOST PROBABLE NEXT STATE
HD	HI
LD	LD,HI
I	X
SI	HD,HI
HI	HD

Figure : Most Probable Next State value for 1^{st} orderMarkov Model (Train Period: 2010 - 2012)[X: Lack of Transition]

CURRENT STATE	MOST PROBABLE NEXT STATE
HD HD	HI
HD LD	SI,HI
HD I	X
HD SI	HI
HD HI	HD,HI
LD HD	X
LD LD	LD,HI
LD I	X
LD SI	HD
LD HI	HD
I HD	X
HI LD	X
I I	X
I SI	X
I HI	X
SI HD	HI
SI LD	X
SI I	X
SI SI	X
SI HI	HD
HI HD	HD,HI
HI LD	LD,HI
HI I	X
HI SI	X
HI HI	HD

Figure : Most Probable Next State value 2^{nd} orderMarkov Model (Train Period: 2010 - 2012)[X: Lack of Transition]

Interpretation and Analysis of Transition Matrices

The transition matrix we got from the Markov model can be interpreted to give some valuable information which is in conformance with practical open source software development.

As shown in figure 9, we get from the 1st order transition probability matrix that every state except HI, has the maximum probability to go to HI. Because as the open source software grows with time, more number of components are being added and no of contributors also increases rapidly. The no of commits also increases. This will lead to huge increase in bug numbers. The interesting thing is that as we reach HI, the most probable next state is HD. This is due to the fact that as the bug number increases above some peak value, huge bug fixing is done to reduce it to decrease the bug numbers. This process continues until software achieves mature phase. As we have trained the data from 2010 to 2013 we observe that absence of I (Insignificant) states. They only occur in initial phase.

As shown in figure 10, in the 2nd order Markov model we also found that from out of 12 feasible transition 8 transitions gives most probable next state as HI. This is also due to huge increase in commits and no of contributors. When bug increases it is also followed by huge bug fixing which intern decreases the bug numbers. Here, also we get some infeasible transitions due to short training period.

5 Performance Evaluation

To evaluate the performance of the and effectiveness of markov model, we have used a formulae to measure our prediction result. The prediction accuracy r is given by :

$r = 1/N - n \times \sum_{t=n+1}^{N} a_t \times 100\%$, Here N is the length of Data Sequence

$$a_t = \begin{cases} 1 \text{ if } x_t = x_t \\ 0 \text{ otherwise} \end{cases}$$

x_t represents the expected next state from the model.

x_t represents the observed next state from the model. 'N ' represents the total no of instances for a particular period and 'n' represents the total number of training instances. 'N-n' represents the no of test instances. During testing if the predicted state and observed state are same then it will be counted as a True(T) and we increase the counter by one. Otherwise, It will be counted as False(F) and counter remains same. The performance is the ratio of number of successful transitions to the total number of transition.

6 Results and Interpretation

To evaluate the performance of our model we tested it upon different no of training instances .We have also evaluated the performance of higher order Markov chain on the same training instances. Then we have also evaluated the performance based on the size of the prediction window. The prediction window size may be 12(Yearly) or 24(Bi-yearly).

6.1 Evaluation based on no of Training Instances

We have trained the Markov model for different training window size like 24,36,48,60, 72. The table 2 shows the training set a test set and accuracy infromation.

Table : Table Showing Training set, Test set and Accuracy Information

Sl No:	Training Set	Test Set	Accuracy
1	2011, 2012	2013(Up to Nov)	60%
2	2010, 2011, 2012	2013(Up to Nov)	60%
3	2009, 2010, 2011,2012	2013(Up to Nov)	60%
4	2008, 2009, 2010, 2011, 2012	2013(Up to Nov)	60%
5	2007, 2008, 2009, 2010, 2011, 2012	2013(Up to Nov)	40%

We found that for small size of training window though accuracy remain same, in the transition matrix there are more missing transitions. So this does not provide complete information about all the transitions. Also we observe that increasing the training window size too much (more than 5 years) also reduces the accuracy. It is due to the fact that there is change in trend in bug patterns with time which intern changes the distribution of states.

Figure 11 shows the most probable state from a given state for the Markov model trained on training set 2. Figure 13 show the prediction results for test set given in table.

Current State	Next State	Prediction Result(T/F)
HI	HD	T
HD	HI	T
HI	HI	F
HI	HI	F
HI	HD	T
HD	HI	T
HI	HD	T
HD	HD	F
HD	HI	T
HI	HI	F

Figure : Prediction Result(1^{st} order Markov Model)

6.2 Evaluation based on Higher Order Markov Chain

We have also trained the 2^{nd} orderMarkov model for different training window size like 24,36,48,60, 72. The table 3 shows the training set a test set and accuracy infromation foe 2_{nd} order Markov model.

Table : Table Showing Training set, Test set and Accuracy Information

Sl No:	Training Set	Test Set	Accuracy
1	2011,2012	2013(Up to Nov)	88 %
2	2010,2011,2012	2013 (Up to Nov)	88%
3	2009,2010,2011,2012	2013 (Up to Nov)	88%
4	2008,2009,2010,2011,2012	2013 (Up to Nov)	88%
5	2007,2008,2009,2010,2011,2012	2013 (Up to Nov)	55%

For second order we get better accuracy than 1st order Markov chain. Figure 12 shows the most probable state from a given state for the Markov model trained on training set 2. Figure 14 show the prediction results for test set given in table 3. As discussed earlier the 2nd order Markov model is converted to 1st order and then transition probability are calculated. As shown in the table, *P* (*HDHD−>HDHI*) is equivalent to *P* (*HDHD−>HI*).

Current State	Next State	Equivalent Next State	Prediction Result(T/F)
HI HD	HD HI	HI	T
HD HI	HI HI	HI	T
HI HI	HI HI	HI	F
HI HI	HI HD	HD	T
HI HD	HD HI	HI	T
HD HI	HI HD	HD	T
HI HD	HD HD	HD	T
HD HD	HD HI	HI	T
HD HI	HI HI	HI	T

Figure : Prediction Result(2_{nd} order Markov Model)

6.3 Evaluation Based upon Size of Prediction Window

We have also tested the performance based on size of the prediction window (Yearly and Bi-yearly). The result of the prediction using 1st order and 2nd order Markov chain is given in Table 4.

Table : Table Showing Training set, Test set and Accuracy Information for different prediction window

Prediction Window Size	Training Set	Test Set	Accuracy(1st)	Accuracy(2nd)
Yearly	2010,2011,2012	2013(up to Nov)	60%	88%
Bi-yearly	2009,2010,2011	2012,2013(up to Nov)	48%	68%

We have also plotted the comparison of predictive performance of the 1st order and second order Markov model on different size of prediction window in figure 15.

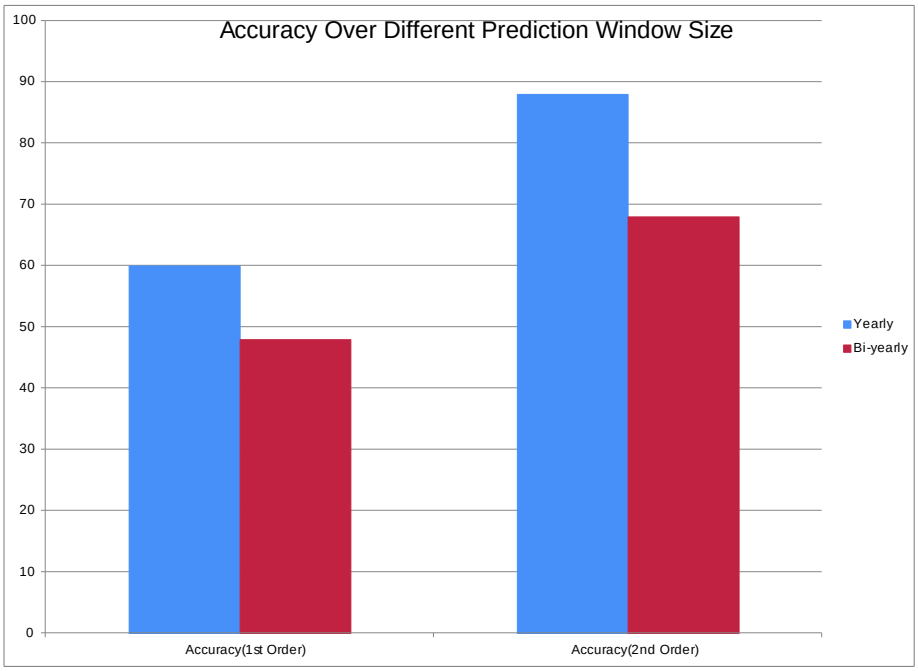


Figure : Prediction Result for Prediction window of different size

The figure shows that the both models give better prediction on prediction window of smaller size. We also observed that 2^{nd} order gives better prediction accuracy than 1^{st} order Markov model.

7. Application of Model

Although the model is demonstrated on Open Source software for which bug information is publicly available, the technique is equally applicable and useful for closed source software also. The model will be useful for software project managers for taking timely decisions such as effort investment and allocation of resources. Another use of this model will be reduction of testing effort as testing team can identify bug growth pattern in advance. The end users by knowing the possible bug patterns in the system in advance can take timely precautions to cope with the loss due to system failure.

8 Conclusion and Future Work

This paper adopts markov chain modeling to predict the software bug growth patterns. The bug data values are represented as 1st order and higher order markov chain for monthly data values. Our results show markov chain as a better predictor of software bug growth patterns. The paper also compares the effectiveness and accuracy of prediction for different types of markov chains and our results confirm 2^{nd} order markov chain as a better predictor of software bug patterns. Our results also show higher prediction accuracy for prediction window of small size in comparison to lager prediction window. In the future We will also apply time series analysis to other open source software systems like mozilla ,eclipse,etc. We wll also check the accuracy in prediction across projects of same and different domains. The goal is to have a generalised model for prediction of temporal bug patterns in software.

References

[1] Mie Mie Thet Thwin; Tong-Seng Quah, "Application of neural network for predicting software development faults using object-oriented design metrics," Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on , vol.5, no., pp.2312,2316 vol.5, 18-22 Nov. 2002.

[2] Rathore, S.S.; Gupta, A., "Investigating object-oriented design metrics to predict fault-proneness of software modules," Software Engineering (CONSEG), 2012 CSI Sixth International Conference on , vol., no., pp.1,10, 5-7 Sept. 2012.

[3] Singh, P.; Verma, S., "Empirical investigation of fault prediction capability of object oriented metrics of open source software," Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on , vol., no., pp.323,327, May 30 2012-June 1 2012.

[4] Cartwright, M.; Shepperd, M., "An empirical investigation of an object-oriented software system," Software Engineering, IEEE Transactions on , vol.26, no.8, pp.786,796, Aug 2000.

[5] Mishra, B.; Shukla, K. K., "Impact of attribute selection on defect proneness prediction in OO software," Computer and Communication Technology (ICCCT), 2011 2nd International Conference on , vol., no., pp.367,372, 15-17 Sept. 2011.

[6] Pai, G.J.; Bechta Dugan, J., "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," Software Engineering, IEEE Transactions on , vol.33, no.10, pp.675,686, Oct. 2007.

[7] Wenjin Wu; Wen Zhang; Ye Yang; Qing Wang, "Time series analysis for bug number prediction," Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on , vol., no., pp.589,596, 23-25 June 2010.

[8] Hongyu Zhang,An initial study of the growth of eclipse defects, MSR '08 Proceedings of the 2008 international working conference on Mining software repositories Pages 141-144 ,ACM New York, NY, USA 2008 .

[9] Bharavi Mishra, K.K. Shukla, Defect Prediction for Object Oriented Software Using Support Vector Based Fuzzy Classification Model, International Journal of Computer Applications (0975 8887) Volume 60 No.15, December 2012.

[10] http://www.tm4.org/*mev*$_m$*anual*/KMC.html

[11] http://www.cra.org/Activities/*craw*$_a$*rchive*/dmp/awards/2003/Mower/KMED.html

[12] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu "Yuanyuan Zhou and Chengxiang ZhaiHave things changed now? : an empirical study of bug characteristics in modern open source software" ASID '06 Proceedings of the 1st workshop on Architectural and system support for improving software dependabilityPages 25 - 33 ,ACM New York, NY, USA ©2006.

[13] Markov Models, Hidden and Otherwise http://kochanski.org/gpk/teaching/0401Oxford.

[14] Dmitrii O. Logofet a,*, Ekaterina V. Lesnaya b, The mathematics of Markov models: what Markov chains can really predict in forest successions, Ecological Modelling 126 (2000) 285–298,Elsevier.

[15] A. Shamshad ,M.A. Bawadi, W.M.A. Wan Hussin, T.A. Majid, S.A.M. Sanusi "First and second order Markov chain models for synthetic generation of wind speed time series" , Energy Volume 30, Issue 5, April 2005, Pages 693–708, Elsevier.

[16] [*web*:*reg*]*http*://*www.web−reg.de/adf*$_a$*ddin.html*

[17] Debian Bug Database, *http*://*udd.debian.org*/*bugs*.

