RT&A, No 2 (68)
Volume 17, June 2022

RELEASE TIME ANALYSIS OF OPEN SOURCE SOFTWARE USING ENTROPY AND RELIABILITY

Vishal Pradhan, Gunjan Tripathi, Ajay Kumar AND Joydip Dhar

•

Department of Applied Sciences, ABV-IIITM Gwalior,
Gwalior-474015, MP, India
vishal.iiitmg@gmail.com, gunjantripathi7@gmail.com, ajayfma@iiitm.ac.in, jdhar@iiitm.ac.in

Abstract

Any software system, however securely written or precise the code is, is always susceptible to failure. These factors, such as the number of errors in the program or the mean-time for software failure, measure the program's reliability. In order to meet more customer needs, current OSS products must be reliable. To measure these parameters, like the reliability of the software, we use different growth models called Software Reliability Growth Models. These models help us in determining the different reliability measures. Faults occur due to several reasons in software- sometimes, it is the environmental factors. It can also be because of casual human behavior. Faults may also occur during the process of removal of previous faults. Whenever the code is changed, randomness in the software increases. We can calculate the optimal release time of a software product based on the calculated reliability measures, which have entropy also been considered. Finally, the user's satisfaction level can also be considered.

Keywords: Entropy, Debugging, Feature improvement, Feature addition, Optimal release time, Software repositories

1. Introduction

The whole idea behind development in the open-source domain is that we initially developed the software's basic model. It can be done by a single individual or a group of developers. After developing the basic software, the code is made publicly available so that people can make changes to it to add new features or make improvements in the current features. People also help in making the software famous amongst the developer community. Code changes can be done by anyone residing remotely in any part of the world. The person can simply request the owner of the software to suggest some new features or modify existing features of the software.

Source code is then again redistributed among the community of developers to cross-check and verify the changes made in the code. Open source software (OSS) is released based on two different methods which decide when the next release is to be scheduled:

1. **Time-based strategy:** In the time-based strategy, specific dates are predicted for the releases and different versions of the software are released on those dates only.

2. **Feature-based strategy:** In the feature-based strategy, new versions are released after implementing certain set of features. New features may be added to the software, or previous features may be modified to increase the satisfaction of the user community.

OSS has a number of characteristics that distinguish it from typical closed source software development.

1.1. Entropy in OSS

Changes are made to code to improve the functionality of the system, add new features, or make modifications to features suggested by the users and developers. When these changes are included, the uncertainty of the software increases. To calculate the increased randomness, information theory can be used as suggested in [10]. This idea of entropy calculation given in the information theory helps us in determining randomness increased in the software.

In this work, the effort is made to calculate the randomness in terms of the entropy in a system by assuming that the randomness increases when the source code is changed for modifications and improvements as discussed in [16]. There are various reasons for making changes to the code. Sometimes the user is not satisfied with the performance due to performance bugs. In the open-source community, since innovators are involved in the development of the software, it is important to keep their interest alive in the product. For this reason, releases are done more often in the open-source products so that contributors can relate to the product and keep making changes to it. The product remains relevant in the market.

Any software goes through a process of evolution in which bugs are fixed, new features are introduced and some features are modified according to the user's satisfaction level. Using this calculated randomness, we can also determine when the next version of the software can come into the market. In this work, non-homogeneous Poisson process (NHPP) based models are discussed. In the software testing process, it is commonly assumed that the cumulative number of failures follows the NHPP and these models are believed to be a reliable model in performance analysis for software [2, 5, 13]. Various NHPP based reliability models for OSS have been developed in recent years [10, 12].

There are two types of environment discussed in the debugging domain [7]. No new bugs are introduced in the perfect debugging model while eradicating the previous bugs. However, in the imperfect debugging model, new errors may get introduced while removing previous ones. Any software goes through a process of evolution in which bugs are fixed, new features are introduced and some features are modified according to the user's satisfaction level.

Whenever bugs are fixed, new features are introduced and modifications are made to the software, it gets better and that is what the motive is. Any organization's goal is to make its software relevant and updated as new changes keep coming into the market. Evolution is a part of any growth process. It is the same for the software industry also. An effort has been made to develop a mathematical model through which we can predict the total number of bugs to be fixed in the future and new features to be added and features to be modified. Since it is not possible to fix all the bugs in a single release, this process continues over the product's whole life cycle.

The objectives are to calculate the entropy of the software using code changes done in the different files of the system and propose a model to determine the total issues based on the calculated entropy and consider the user's satisfaction level. Also, make the prediction of release time of the next version based on entropy which the product manager can use to reduce the overall cost of the product.

2. Literature review

For any software to remain relevant in the market, it needs to grow with the user's expectations and new technologies coming into the market. Changes are a necessity for the growth and development of the product as in [19]. When new changes are made in the code, the randomness in the software increases.

Previous research has been done in this direction in considering this entropy as a measure in the calculation of release date of the software [6, 11, 15, 17, 18]. When the entropy increases, the software becomes more complex with time. This research has been made to quantify these changes and calculate the randomness that occurred because of those changes. After the entropy is calculated through file changes, the total number of issues to be fixed, including the bugs, newly introduced features, and feature improvements, are predicted. In the end, an effort is also put to predict the next release time of the version based on the entropy calculation.

In the past, numerous efforts have been made in the direction of measuring the reliability of the software and thus decreasing the overall software assessment budget of the product [10]. In most of the earlier models, only two factors are mostly considered in the software design process-reliability and the cost of the product. These are inversely proportional to each other. If we wish to increase the reliability of the software, we need to gice more time to it which increases the cost or if we try to decrease the cost, we compromise with the reliability of the software [3]. Many of the models used for prediction in the closed source projects give too optimistic results in the open source domain [8, 9].

In the models used for prediction in the open source domain, testing efforts are considered and time taken to correct the faults are also measured [3]. Dai et al. [4] put an effort to find out the randomness in the reliability modelling of a single component within a large software and attributes are also assumed to be correlated. Authors have also taken into consideration, the views of the subject expert and the historical change data of the software.

Kamavaram and Goseva-Popstojanova [8] performed the research based on including entropy for the calculation of software reliability engineering by using it in the Markov model which is used for software specifications. Randomness of the operational profile is calculated and the model proposed is architecture based. effort is also made to introduce the concept of conditional entropy. Kerzazi and Khomh [9] in this research considers one of the most important thing in release engineering, the time for a software code to reach from the development environment to the production environment considering the quality analysis at each stage. Only data from the actual industrial organizations are taken over a long period of time (15 months) and around 250 releases are taken into consideration.

Li et al. [10] investigated one of the left out factors in open source software- reliability. Two important parameters which are stressed upon are the fast release of software and the reliability. Though these are contradicting in nature but they are most important factors to b considered. Release planning model is presented through this research. Multi attribute theory is also considered. Michlmayr et al.[11] focused on the time-based release model of OSS. In the time-based model, new releases come in the market after a certain pre-decided interval of time. They have taken interviews of members from seven open source organizations with volunteer workers and have analyzed the benefits of time-based release models.

Ruhe [14] researched thoroughly on the tools and techniques through which products can be made in a manner such that resources are utilized efficiently and users expectations can also be met by the product. Methods to build successful product are discussed. Release planning problem is also discussed.

3. Objectives

Any software goes through a process of evolution in which bugs are fixed, new features are introduced and some features are modified according to the user's satisfaction level.

Also, certain issues are left unnoticed, which the innovators in future development processes remove. This can happen due to correction delay. When the life cycle process of the development starts, the issues which were not fixed in previous releases are also taken into consideration and are added to the issue content of the current work. The objectives are:

- Calculate the entropy of the software using code changes done in the different files of the system.
- Propose a model to determine the total issues based on the calculated entropy and consider the user's satisfaction level.
- Prediction of release time of the next version is based on entropy which the product manager can reduce the product's overall cost.

4. Research Methodology

4.1. Dataset

The dataset collection process involved collecting issues from the issues directory of different Apache products. Products have multiple releases and issues are fixed before each release.

Multiple open-source projects have the details of their issue available on multiple platforms like Bugzilla. Apache projects are big projects and they involve a large number of contributors involved in the development of the products. Different products of Apache open source are considered, for example, Avro, jUDDI and Hive. All the fixed issues - either bugs, new features, or improvised features are downloaded from the issue tracking repository available on the Bugzilla website.

4.2. Methodology

After collecting data of issues from the issues directory, release dates are also noted from the git-hub repository. Then the date of fixing issues is mapped with the release date of products. Then, monthly entropy can be calculated according to the number of changes made in different projects and modules of the product. After the calculation of entropy, this is mapped with the release date noted from the git-hub repository. In the calculation of entropy, the code change process is termed as an event and noted down. Here, an event is defined as the process during which the code of a software is changed.

First, all the issues, including the bugs, new features introduced or suggested by the users of the software and feature improvements, are downloaded from the issue tracking repository. All product details with issues are available on the issues.apache.org website. Then for each product, their bugs, feature additions and feature improvements are calculated on a month-to-month basis. We also made a note of the date when these issues were fixed.

The release date of the products can be extracted from the git-hub using the git-hub tool. Whenever changes are made in the source code and committed, they are being done to fix some issues or add some new features. Git-hub tool is easily available on the git-hub site.

4.3. Implementation

Issues are defined as the bugs present in the software, new features to be added and the feature modifications to be performed. There are two types of contributors, innovators and imitators. Innovators fix the issues at rate 'p' and imitators fix issues at rate 'q'. We assume that the total number of issues identified as the target issues to be fixed in a particular release is constant and represented by 'a'. Initially, since no issues are fixed, we can represent this condition with a differential equation as follows:

$$\frac{d(X(t))}{dt} = p(a - X(t)) + q\frac{X(t)}{a}(a - X(t)) \tag{1}$$

where X(t) represents the value of fixed bugs cumulatively.

Assuming the initial conditions at t = 0 , no bugs are fixed, that is X(0) = 0, we have:

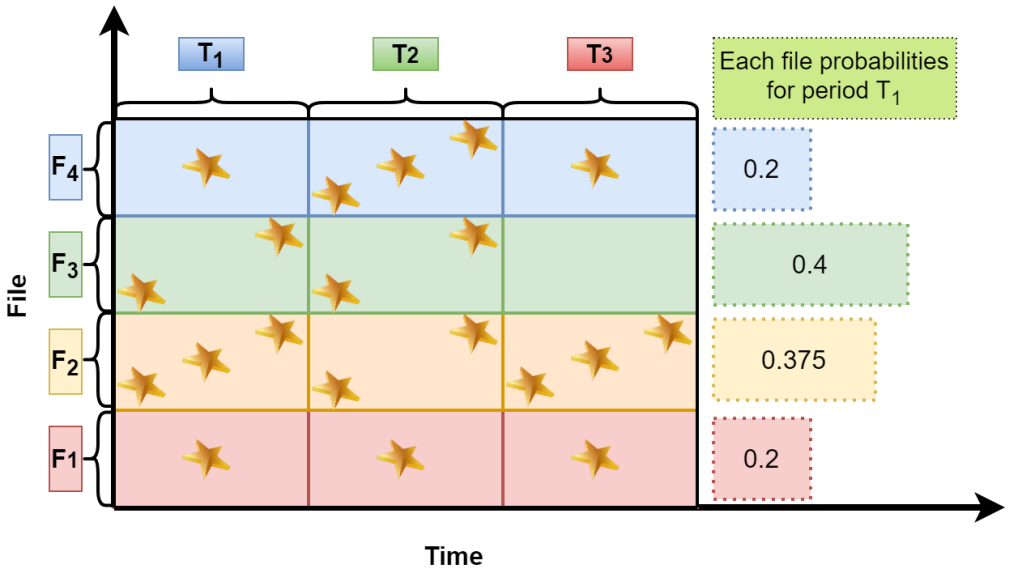
$$X(t) = a\left[\frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p}e^{(-(p+q)t)}}\right] \tag{2}$$

Here, q/p remains constant and p+q represent the rate at which issues are fixed per remaining issue.

5. Entropy calculation

To calculate entropy, we must first find out the amount of information in the software code. Information theory can be used to find that. For our model, we define data as the event in which a file is changed for modification in the code. This can be used to calculate the amount of randomness associated with the software system.

We have considered four files and the time interval during which changes were made in those files is noted down. Where p gives the probability that the file will be changed during that period, we calculated it by dividing the number of times this specific file is changed in that period by the total number of changes made in all the files. This is shown in the figure.



There are multiple reasons for changing the code of the software. When users suggest bugs and new user requirements come up, developers make changes in the code to modify modules or add new modules. Sometimes there are logical errors present that also need to be rectified. Using the Cobb-Douglas equation, we can include both time and entropy for the calculation of output as follows, where time is represented by 's' and entropy is represented by 'u'.

$$t = s^{\alpha} u^{(1-\alpha)} \tag{3}$$

where $0 <= \alpha <= 1$

Using the Cobb-Douglas function, we can integrate time and entropy in the calculation of X(t) to observe the effect of both time and entropy. The model to calculate issues fixed can thus be represented as having both entropy and time-integrated. The model can thus be represented as:

$$X(s,u) = a\left[\frac{1 - e^{-b(s^{\alpha}u^{(1-\alpha)})}}{1 + \beta e^{-b(s^{\alpha}u^{(1-\alpha)})}}\right] \tag{4}$$

In most of the OSSs, all issues are not fixed in the same release. Some are passed on to the next release. These issues are then added to the content of issues for a particular version and then fixed by the developer team.

5.1. Code History Metric

Code History Metric(CHM) is a method with which we calculate the complexity of code changes in the software [1]. The whole concept of code changes is used for the calculation of CHM. We

also measure one more factor, which is Code History Period Factor(CHPF), for a file j during a time period of i as:

$$CHPF(i,j) = C_{ij}H_i where j \in F_i \tag{5}$$

Here, H_i is the entropy calculated for the changes done in the file during a period of time interval i and C_{ij} represents the contribution of entropy for the given file j. We are considering several cases here:

Case 1 : If C_{ij} = 1, we assume full complexity and all files having the same weight during this interval.

Case 2 : If $C_{ij} = P_j$, we assume contribution is equal to the probability of file j being changed in the time interval.

5.2. Releasing Versions of Software

Steps for calculating the matrix are as follows:

1. First a note of the release date is taken for every version of the software.
2. Date on which each file was changed is also noted.
3. Classification of bugs fixed in the files as a new feature, feature improvement and feature modification.
4. Calculate the total number of bugs fixed.
5. Record the issues in each released version of the software.
6. Arrange the changes according to the month in which they were made and the matrix is calculated.
7. For every release, the time at which the software was released is found.

Though making new changes to the software is a requirement for every organization, it brings new faults into the system. The code becomes more complex over time and it becomes challenging to keep it relevant and reliable at the same time. Predicting the modules that have more number of bugs than the others is beneficial in the sense that then the product manager can decide how to allocate resources among the parts of the software such that an efficient version is released with more reliability and user satisfaction level.

Predicting bugs also helps in reducing the testing time of the team and thus, the overall expenditure can be reduced.

Based on the change of code of the software, bugs and release time are predicted. As a part of the previous work, a few previous models are considered and their results are compared with the proposed model in the next section.

6. Experiments and results

6.0.1 Experiments

Assuming that initially no issues are there and issues will occur in further releases only after suggestions from the users come, at t = 0, X(t) = 0. This gives the following equation :

$$X(t) = a\left[\frac{1 - e^{-bt}}{1 + \beta e^{-bt}}\right] \tag{6}$$

Here, value of q/p does not change with time and p+q is the rate at which issues are fixed per remaining issues.

Rate at which cumulative number of issues are fixed is given by:

$$\frac{dX(t)}{dt} = \frac{f(t)}{1 - F(t)}(a - X(t)) \tag{7}$$

Here, F(t) is the distribution function representing the iussues to be fixed anf f(t) is the density function, relation between them is f(t) = d F(t)/dt.

At any given time t, (a - X(t)) represents the number of issues yet to be fixed in the code. As time increases, more issues are fixed and (a - X(t)), i.e, remaining issues in the software decreases with time. This in turn increases the reliability of the software[4].

For a particular software product, 'Ai' is the real number of issues to be fixed, including the bugs, new features to be added and the modifications to be performed in the software. Where 'ai' is the potential issues that need to be fixed. 'ai-Ai' represents the issues left in the previous release and are now in the issue content of the current release. Fixing efficiency is denoted by 'bi'.

Comparing the results of our model with the JM and S-shaped mode, we get the results as shown in the table 1.

Table 1: *Comparison Table*

Release Number	Model	a	b	Ai	ai-Ai
1	JM Model	360	0.121		28
1	S-shaped Model	621	0.173		289
1	Proposed Model	364	0.527	332	32
2	JM Model	197	0.13		14
2	S-shaped Model	221	0.268		38
2	Proposed Model	507	0.076	183	324
3	JM Model	264	0.078		55
3	S-shaped Model	300	0.174		91
3	Proposed Model	233	0.3	209	24
4	JM Model	219	0.063		77
4	S-shaped Model	181	0.211		38
4	Proposed Model	210	0.173	142	9
5	JM Model	85	0.116		10
5	S-shaped Model	104	0.233		9
5	Proposed Model	75	0.743	75	0

The release time of the software is calculated using linear regression in multiple stages. Assuming that the time, p_0 is a dependent variable and the Code History Matrix (x_0) and the number of bugs ($x1$) are independent variables, time can be calculated as:

$$p_o = a_0 + a_1 x_0 + a_2 x_1 \tag{8}$$

Here, a_0 and a_1 are regression coefficients whose value can be found out by linear regression method.

Calculated values of time to release from the Code History Matrix are also shown in the table 2.

Table 2: *CHM Table*

Total Changes	CHM(1)	CHM(2)	Time
673	7.288	2.228	2.248
286	2.335	0.499	2.723
1626	8.931	3.405	2.726
774	10.07	3.056	3.934
738	4.034	0.901	2.467
911	6.545	1.698	3.303
1524	8.361	2.311	2.812
302	3.296	0.804	3.421
2414	10.19	2.522	5.074
2331	3.535	0.783	3.058
943	5.795	1.374	3.317
576	6.463	1.593	2.562
413	3.456	0.886	1.143
382	5.178	1.32	3.335
1883	5.302	1.422	1.895
234	1.757	0.309	0.663
500	2.324	1.273	1.086
898	5.361	1.256	4.918
1254	5.252	1.978	3.337
827	7.04	0.256	2.443
342	1.629	1.304	0.661
734	5.377	1.543	2.803
891	5.464	1.064	3.007
1353	4.795	1.476	1.412
850	6.447	0.143	2.461
1135	1.578	0.214	1.128
334	1.645	0.583	1.143

7. Results and discussion

The whole idea behind OSS development is that we initially developed the software's basic model. It can be done by a single individual or a group of developers. After developing the basic software, the code is made publicly available so that people can make changes to it to add new features or make improvements in the current features.

The aptitude of the testing team also determines the number of faults introduced into a software. Environmental factors such as different operating system specifications can also contribute to introducing bugs into the software. Whenever developers try to remove a bug from the code, they make changes to the software. These changes can be quantified in terms of entropy. The time of the next release is helpful both to the product manager and the user of the software.

7.1. Ease of Testing

One of the main stages of the software development process is the testing phase. This model helps to ease the process in the testing phase by providing a direction to the testers so that they can focus their energy and resources in one direction and get better and more efficient results. Once the cost of the testing process is minimized, the whole development process cost is reduced.

7.2. Managerial implications

Product managers can use this approach to find out the best time to release the software in the market. Rapid release strategy and reliability are conflicting parameters and it is crucial to balance between them. Product managers can look at the future trend of bugs and plan their releases accordingly.

8. Conclusion

There is always scope for improvement. Although the proposed model helps determine the important parameters of modelling, the manager's decision is still subjective. There are still many more different factors in different projects that matter when product release is considered. So our model can be used to make a general idea about the situation, but the final decision needs to be taken considering many other factors also.

Other factors that need to be considered are past data of the product and historical experiences of the team. Only then a more trusted decision can be taken. Efforts should also be made to study more closed source projects and compare them to the calculation of different parameters.

One more limitation is that calculation of entropy based on the files changed is done manually. This process can also be made more precise and accurate by using other methods of entropy calculation if available. More research can be done in this direction.

References

[1] Anand, A., Bharmoria, S. and Ram, M., 2019, Characterizing the Complexity of Code Changes in Open Source Software, *In Recent Advancements in Software Reliability Assurance*, Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742: CRC Press, pp. 1–14.

[2] Anand, A., Kaur, J., Singh, O. and Ram, M., 2021, Optimal Resource Allocation for Software Development under Agile Framework, *Reliability: Theory & Applications*, (SI 2 (64)), pp. 48–58.

[3] Cathedral, R.: 2001, the bazaar raymond es the cathedral and the bazaar: Musings on linux and open source by an accidental revolutionary.

[4] Dai, Y.-S., Xie, M., Long, Q. and Ng, S.-H.: 2007, Uncertainty analysis in software reliability modeling by bayesian analysis with maximum-entropy principle, *IEEE Transactions on Software Engineering* **33**(11), 781–795.

[5] Dhaka, R., Pachauri, B. and Jain, A., 2021, Two-Dimensional SRGM with Delay in Debugging by Considering the Uncertainty Factor and Predictive Analysis, *Reliability: Theory & Applications*, (SI 2 (64)), pp .82–94.

[6] Gacek, C. and Arief, B.: 2004, The many meanings of open source, *IEEE software* **21**(1), 34–40.

[7] Hassan, A. E.: 2009, Predicting faults using the complexity of code changes, *2009 IEEE 31st international conference on software engineering*, IEEE, pp. 78–88.

[8] Kamavaram, S. and Goseva-Popstojanova, K.: 2002, Entropy as a measure of uncertainty in software reliability, *13th Int'l Symp. Software Reliability Engineering*, pp. 209–210.

[9] Kerzazi, N. and Khomh, F.: 2014, Factors impacting software release engineering: A longitudinal study, *Proc. 2nd Workshop Release Eng*, pp. 1–5.

[10] Li, X., Li, Y. F., Xie, M. and Ng, S. H.: 2011, Reliability analysis and optimal version-updating for open source software, *Information and Software Technology* **53**(9), 929–936.

[11] Michlmayr, M., Fitzgerald, B. and Stol, K.-J.: 2015, Why and how should open source projects adopt time-based releases?, *IEEE Software* **32**(2), 55–63.

[12] Pradhan, V., Kumar, A. and Dhar, J., 2022, Modeling Multi-Release Open Source Software Reliability Growth Process with Generalized Modified Weibull Distribution, *Evolving Software Processes: Trends and Future Directions*, pp. 123–133.

[13] Pradhan, V., Kumar, A. and Dhar, J. 2022, Modelling software reliability growth through generalized inflection S-shaped fault reduction factor and optimal release time, *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 236(1), pp. 18–36.

[14] Ruhe, G.: 2010, *Product release planning: methods, tools and applications*, Auerbach Publications.
[15] Saxena, P., Kumar, V. and Ram, M., 2021, Ranking of Software Reliability Growth Models: A Entropy-ELECTRE Hybrid Approach, *Reliability: Theory & Applications*, (SI 2 (64)), pp. 95–113.
[16] Singh, V., Sharma, M. and Pham, H.: 2017, Entropy based software reliability analysis of multi-version open source software, *IEEE Transactions on Software Engineering* **44**(12), 1207–1223.
[17] Spinellis, D.: 2015, The strategic importance of release engineering, *IEEE software* **32**(2), 3–5.
[18] Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S. B. and Shafique, M. U.: 2010, A systematic review on strategic release planning models, *Information and software technology* **52**(3), 237–248.
[19] Wright, H. K. and Perry, D. E.: 2012, Release engineering practices and pitfalls, *Proceedings of the 34th International Conference on Software Engineering*, IEEE Press, pp. 1281–1284.