Predicting bug-fixing time: A replication study using an open source software project

The Journal of Systems and Software – 2017

Akshay Ravichandran
North Carolina State University
aravich6@ncsu.edu

ABSTRACT

On projects with tight schedules and limited budgets, it may not be possible to resolve all known bugs before the next release. Estimates of the time required to fix known bugs (the “bug-fixing time”) would assist managers in allocating bug fixing resources when faced with a high volume of bug reports. In this replication study, Akbarinasaji et. al [1] attempt to validate the generalizability of the original work done by Zhang et. al [2] by applying their methods to open source data from Bugzilla Firefox [3] and predicting bug-fixing time.

KEYWORDS

Markov model; Monte Carlo simulation; KNN classification

INTRODUCTION

Software bugs are an inextricable part of software maintenance and development activities. A significant amount of time is spent by software developers in investigating and fixing bug reports. The motivation behind the original study and this replication study is that having advance knowledge of bug fixing time would allow software quality teams to prioritize their work.

The research questions addressed by both sets of authors are:

- **RQ1:** How many bugs can be fixed in a given time?
- **RQ2:** How long would it take to fix a given number of bugs?
- **RQ3:** How long would it take to fix an individual bug?

The methods proposed by the original study proved successful when applied to bug data collected from three commercial projects from CA technologies. This study aims to apply the same methods to an open source tool to verify their generalizability to all software systems.

RQ1: HOW MANY BUGS CAN BE FIXED IN A GIVEN TIME?

Zhang. et al proposed the use of a **Markov-based model**. The authors of this paper have also adopted the same method, however the number of states a bug can be in is 8 instead of 4. Figure 1 depicts the Markov model for a bug’s life cycle in Bugzilla. Each of these transitions have a probability attached to them, which can be calculated based on historical bug-resolution data. The probabilities can be represented in the form of a matrix as given in Figure 2.

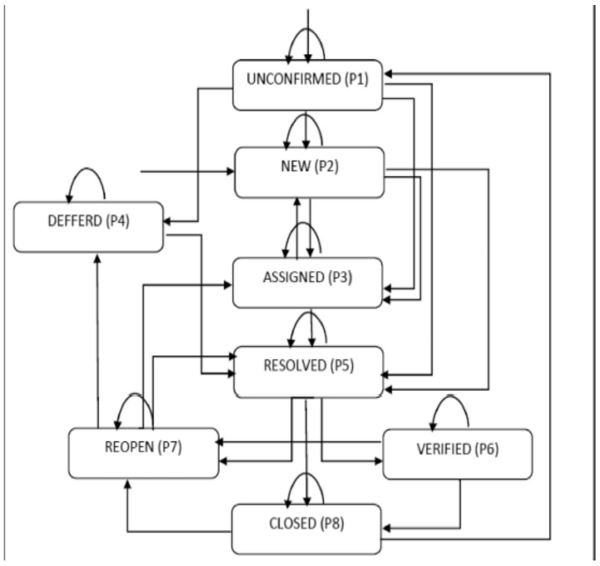

Figure 1: Markov model of bug life cycle in Bugzilla

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & \cdots & P_{18} \\ P_{21} & P_{22} & \cdots & \cdots & P_{28} \\ P_{31} & P_{32} & \cdots & \cdots & P_{38} \\ \cdots & & & & \\ P_{81} & P_{82} & P_{83} & \cdots & P_{88} \end{bmatrix}$$

Figure 2: State transition probability matrix

Here, $\mathbf{P_{ij}}$ represents the conditional probability that a bug will be in state “j” at time t+1, given that it is in state “i” at time t. The training period for calculating the transition matrix was 12 months, and it was used to predict the number of fixed bugs after 1 month, 2 months and 3 months following the training period.

Table 1 shows the results of the Markov based prediction model. As we can see, the predicted value is very close to the actual value in all 3 months for the Bugzilla project. The MRE values are lower than those for the projects in CA Technologies, thus the proposed model is a very good fit for finding the number of bugs that can be fixed in a given time.

	Period	Predicted	Actual	MRE
Project A	Month 1	18.38%	18.91%	2.831%
Original study	Month 2	19.81%	21.30%	6.997%
	Month 3	21.00%	23.78%	11.686%
Project B	Month 1	24.70%	25.90%	4.633%
Original study	Month 2	26.60%	27.10%	1.845%
	Month 3	28.20%	29.20%	3.425%
Project C	Month 1	18.84%	18.64%	1.070%
Original study	Month 2	20.14%	20.44%	0.015%
	Month 3	21.24%	21.04%	0.948%
Firefox	Month 1	70.50%	71.04%	0.766%
Replicated study	Month 2	69.61%	70.72%	1.579%
	Month 3	68.38%	67.27%	2.745%

Table 1: Results of Markov based prediction model

RQ2: HOW LONG WILL IT TAKE TO FIX N BUGS?

Zhang et. al proposed the use of **Monte Carlo simulations**. Because bug fixing time was skewed, they employed the lognormal, exponential and Weibull distributions and picked the distribution with the highest **R^2** value and lowest **S_e** value. After this, they randomly picked N samples from the distribution, summed and saved them. This was repeated many times and the average was taken as the time needed to fix N bugs. The same approach was used by the authors of this paper. However, they filtered their data to exclude non-deferred bugs because of their future research interests. For Bugzilla, the proposed method outperforms the average-based method by a large factor.

	Distribution	R^2	S_e
Firefox	Exponential	0.4963	8.9330
Replicated study	Lognormal	0.9998	0.1777
	Weibull	0.8817	4.327

Table 2: R^2 and S_e values for different distributions

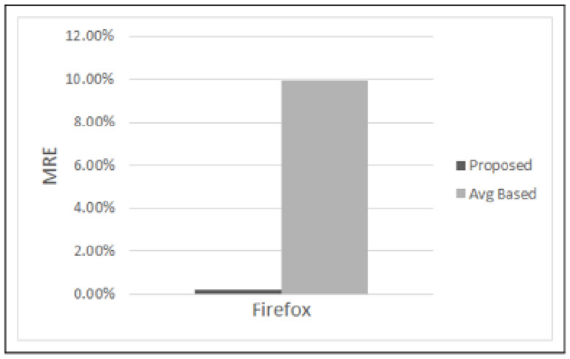

Figure 3: Comparing Monte Carlo simulations with avg-based method

RQ3: HOW MUCH TIME WILL IT TAKE TO FIX A GIVEN BUG?

Zhang et. al proposed the **K nearest neighbor classification**. The features used in this replication study include submitter, owner, priority, severity, component and summary. The bugs were classified as **slow** or **quick** fix based on different time thresholds. Table 3 shows the comparison between the F values of the existing methods and the proposed method. As we can see, in general the proposed method performs better than all existing methods.

	δ	0.1	0.2	0.4	1	2
Project A	Proposed	0.704	0.659	0.668	0.728	0.852
Original study	BayesNet	0.702	0.633	0.638	0.704	0.828
	NaiveBayes	0.701	0.636	0.638	0.698	0.832
	RBFNetwork	0.691	0.631	0.630	0.696	0.833
	Decision Tree	0.653	0.495	0.643	0.601	0.798
Project B	Proposed	0.679	0.685	0.650	0.672	0.703
Original study	BayesNet	0.669	0.613	0.597	0.637	0.643
	NaiveBayes	0.670	0.610	0.605	0.642	0.645
	RBFNetwork	0.679	0.614	0.610	0.621	0.628
	Decision Tree	0.618	0.520	0.627	0.670	0.632
Project C	Proposed	0.770	0.793	0.762	0.769	0.773
Original study	BayesNet	0.787	0.768	0.722	0.745	0.780
	NaiveBayes	0.790	0.773	0.739	0.745	0.775
	RBFNetwork	0.784	0.780	0.739	0.741	0.750
	Decision Tree	0.662	0.776	0.741	0.740	0.765
Firefox	Proposed	0.5616	0.6128	0.6102	0.6144	0.7357
Replicated study	BayesNet	0.4967	0.4912	0.5163	0.5495	0.5314
	NaiveBayes	0.4980	0.4993	0.5122	0.5505	0.5701
	RBFNetwork	0.3820	0.3521	0.4425	0.3864	0.5230
	Decision Tree	0.3820	0.3521	0.3542	0.3864	0.6599

Table 3: Comparison of F values

CONCLUSIONS

Based on the results obtained from these experiments, it can be tempting to concluded that the original methods proposed by Zhang et. al can in fact be extended to all open source projects. However, there are threats to this validity of this claim.

1. **Internal Validity**
   Both the original study and this one assumed that the project under consideration is evolving and stable. However, the Markov assumption might fail for short-lived, non-systematic bug handling projects.
2. **External Validity**
   All results are based on data collected from a single open source bug repository, so they reflect the behavior of that specific organization. Thus, it would be incorrect to generalize for all open source projects.

However, despite these threats, the results are still promising. Going forward, these methods can serve as a good baseline for predicting bug-fixing time.

REFERENCES

[1] S Akbarinasaji, B Caglayan, A Bener - Journal of Systems and Software, 2018
[2] Zhang, H., Gong, L., Versteeg, S., 2013. Predicting bug-fixing time: an empirical study of commercial software projects. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp. 1042–1051
[3] Bugzilla https://www.bugzilla.org/