Reliability Growth of Open Source Software using Defect Analysis

Sharifah Mashita SyedMohamad Tom McBride
Faculty of Information Technology, University of Technology, Sydney
{sharifah, mcbride}@it.uts.edu.au

Abstract

We examine two active and popular open source products to observe whether or not open source software has a different defect arrival rate software developed in-house. The evaluation used two common models of reliability growth models; concave and S-shaped and this analysis shows that open source has a different profile of defect arrival. Further investigation indicated that low level design instability is a possible explanation of the different defect growth profile.

Key Words- Open Source Software, Software Reliability, Defect Classification.

1. Introduction

Open source software development has gained popularity around the world. As open source development does not necessarily adhere to traditional software engineering, particularly in the area of formal testing, the quality of open source software needs to be examined. In open source model quality assurance processes are performed in a different manner than traditional software engineering [15].

Reliability is one of the more important characteristics of software quality when considered for commercial use. Adoption of reliable open source products for commercial use can be a real challenge. While open source software products routinely provide information about product activity rank, number of developers and the number of users or downloads, this information does not convey information about the quality of the open source product.

Here we measure open source software product reliability using an approach frequently used in commercial software development. As measuring defect growth is a good empirical way of evaluating software quality [6], we investigate whether or not open source software has a different defect arrival rate compared to in-house developed software. If it does then new tests and models for analysing open source reliability must be developed. If not, then the same tests of product reliability can be applied.

We first describe common models used to measure software reliability. Then we describe defect datasets used in this examination and discuss which reliability growth model is a better fit to the datasets. We then briefly described defect classifications and discuss more results of open source profile before drawing some conclusions and note our further direction.

2. Software reliability analysis models

Reliability growth models, which are statistical analysis of software failure data during development, traditionally deal with prediction after making some fundamental assumptions on the error detection process. The models are essential measurement during the testing phase of software development for examining the degree of reliability, and thus quality, of the developed product.

Attributes of the reliability models have been usually defined with respect to time with four general ways to characterize reliability [11]: time of failure, time interval between failures, cumulative number of faults up to a period of time and failure found in a time interval. A popular model compares the cumulative numbers of fault to cumulative usage time (can be calendar time or execution time) [13; 10].

When plotted over time, most reliability growth models are represented by either the Concave or the S-shaped curve [16]. Goel-Okumoto non-homogeneous Poisson model [8] and Musa model are among the earliest reliability models that show concave growth curve (also called exponential). The Goel-Okumoto model assumes that a software system is subject to failures at random times cause by defects present in the system, thus takes number of defects per unit of time as independent Poisson random variables. Note that Poisson distribution has been found to be an excellent model in many fields of application where interest is in the number of occurrences [7]. The mathematical representation of concave model is:

$$\mu(t) = a(1 - e^{-bt}) \text{ where } a, b > 0$$

and S-shaped growth function is:

$$\mu(t) = a(1 - (1 + bt)e^{-bt}) \text{ where } a, b > 0$$

$\mu(t)$ The expected number of defects occurrences for any time, *t*

a The expected total number of defects to be observed eventually

b The shape factor or defect detection rate per defect

The S-shaped model derives from a modification of the Goel-Okumoto model. The S-shaped curve reflects to the initial learning period at the beginning, as testing people become familiar with the software, followed by growth and then stabilizes as the residual faults become more difficult to discover. Both models apply a same parameter i.e. shape factor.

In this research, we will plot defect data using these two common models and examine whether there is a different in defect arrival between open source software and in-house source software.

3. Data collection

3.1. Defect data

We identified two notable and active open source projects from SourceForge.net (http://sourceforge.net/). We will refer to these projects as Open Source A and Open Source B. These are two of the most successful and widely used among open source communities under different topics or application domain. Both of the chosen projects are considered stable, in production. We collected defect data of the selected projects from the SourceForge.net tracking tool. This captures all of the standard defect attributes that have associations to occurrence date. Table 1 and Table 2 list the information of the projects.

Table 1: Open Source A details

Register date	Defects over the project lifetime	
	Overall	Accepted
Nov 2000	300	130

Table 2: Open Source B details

Register date	Defects over the project lifetime		Defects reported during 2007	
	Overall	Accepted	Overall	Accepted
June 2000	514	362	136	75

Note that we did not analyze the entire defect data but removed from consideration trivial defects such as cosmetic defect, design defects such 'look and feel' and platform configuration defects. We also excluded duplicated defects those invalid defects that had been deleted by the open source project administrator. Defects are given a severity from 1, lowest, to 9, highest with most classified as severity 5. Our data set included only defects with severity 4 or higher in order to achieve a reliable defect profile for the open source software. As can be seen in Table 1 and Table 2, the total number of accepted defects is lower than the overall number of reported defects.

Defect data for a software project developed in house using normal commercial software development processes was collected from an organization in the telecommunications industry. Defect considered were only those that had been discovered and reported by the development team, as opposed to defects reported after release [12]. The in house defect data is maintained in a web-based bug tracking system Again, we perform the same data cleaning activities to the defect data and a summary of the in-house project is shown in Table 3.

Table 3: In-house project details

Defect record since	Overall defects over project life time	Overall defects over year 2007	Accepted defects over year 2007
Sept 2005	926	106	100

4. Analysis and Findings

We analyzed the datasets using SPSS nonlinear regression analysis in which each of the datasets is transformed into two models for reliability analysis.

4.1. Reliability growth models

As mentioned earlier, concave and S-shaped growth models apply the same parameter i.e. shape factor; and this explains the spread of the data. The estimated parameters and R squared are listed in Table 4. Briefly, R squared as an indication of how good the correlation between the cumulative number of defects to the project lifetime.

Table 4: Estimated parameters (b) and R squared of the reliability growth models

	Concave model		S-shaped model	
Projects	*b*	*R squared*	*b*	*R squared*
Open Source A	0.270	0.827	0.570	0.554
Open Source B	0.008	0.641	0.230	0.799
In-house	0.150	0.967	0.315	0.995

We plotted cumulative defects found over the life of open source (Figure 1 and Figure 2) and in-house projects (Figure 3). The observed curves represent the actual values of the projects, the predicted curves correspond to the estimation of concave and S-shaped curves. Obviously, both open source datasets did not converge to the concave and S-shaped curves. The predicted concave and S-shaped curves badly miss on almost all of the observed values.

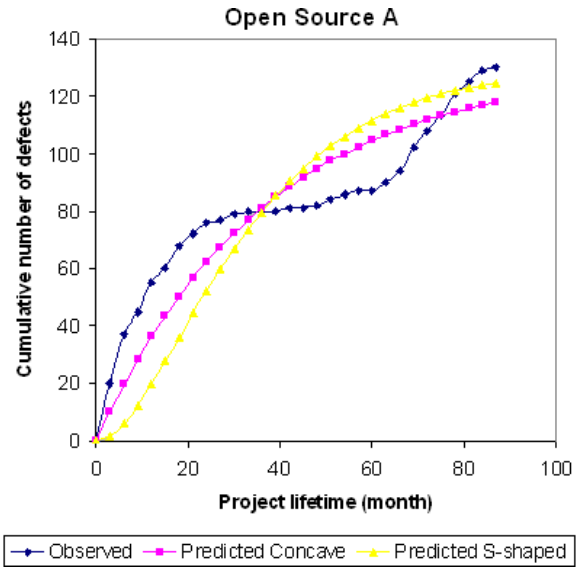

Figure 1: Project A reliability growth curve

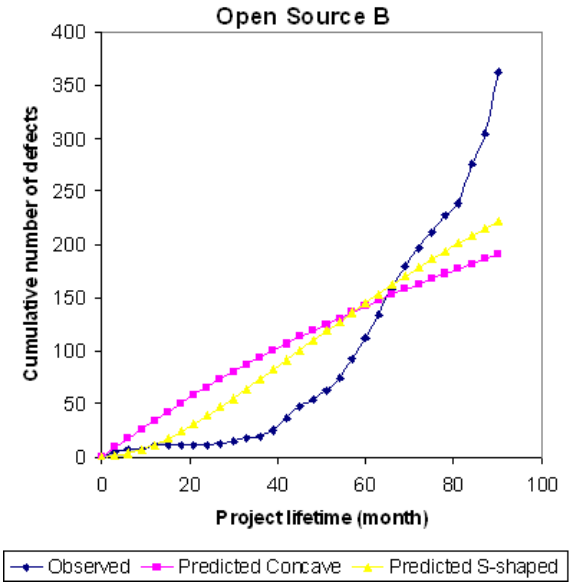

Figure 2: Project B reliability growth curve

Interestingly, the overall growth curve of Open Source A (Figure 1) seems to exhibits a reverse S-shaped curve. It appears as a concave curve only in the first period of the calculated project life span. A different growth pattern can be seen for the Open Source B (Figure 2), whereby the curve shows convex rather than concave shape. As expected, we obtained a good fit model for in-house product (Figure 3). The S-shaped and concave curves did come very close to the observed data.

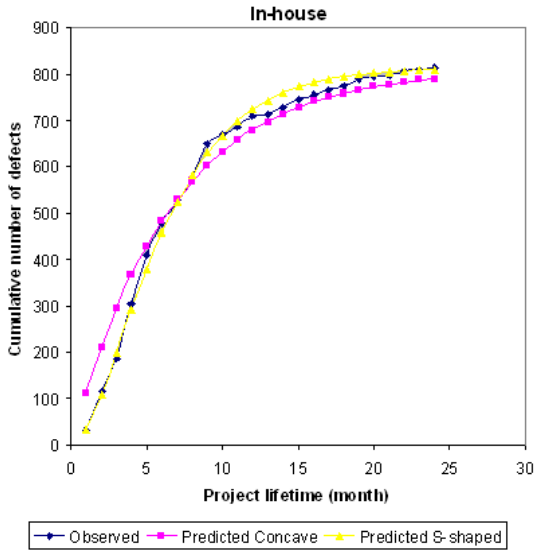

Figure 3: In house project reliability growth curve

Examination of these results Table 4 shows a little confusing of indications of the reliability models. For instance, Open Source A has a high R squared result for the concave model, but that is not the case if we look at the predicted concave curve as illustrated in the graph (Figure 1). In general, both of the open source products did not exhibit either any of the common reliability growth models, thus we conclude that open source software exhibits a different defect arrival rate for the projects examined so far. These findings deserve further investigation on what could contribute to that difference. Here we extend our work to examine the defect types within the overall defect growth.

4.2. Defect classification

Defects can be classified by types. Many studies have shown the use of defect type information as the important means of assessing the relative quality of a software system [2; 14]. IEEE Standard 1044 [1], Hewlett-Packard (HP) defect scheme [9] and Orthogonal Defect Classification (ODC) [3] are among the well known defect classifications. In general, these classifications comprise about the same information of type of defects.

We used ODC defect types [12] to classify the datasets. ODC has demonstrated its value in revealing insights into software quality and software development [5; 4]. Table 5 shows the ODC defect types and its process associations used in this study.

Table 5: The defect type and process associations–[3]

Defect type	Description of defect type	Process Associations
Function/Class/ Object	missing or incorrect functionality	Design
Interface/O-O Messages	affects the interaction of components via macros, call statements and/or parameter lists	Low Level Design
Timing/Serialization	serialization of shared resource is wrong or missing	Low Level Design
Algorithm/Method	efficiency or correctness of an algorithm or local data structure	Low Level Design
Checking	missing or incorrect data validation in conditional statements.	Low Level Design/Code
Assignment/Initialization	values assigned incorrectly or not assigned at all	Code

We manually classified the data and for each defect and assign a qualifier of either missing or incorrect. Accuracy in classifying defects may become an issue although the 'orthogonality' defect classes reduce the probability of misclassification. Examining relationships between the 'type' and 'qualifier' can reveal weaknesses in the explicit areas of software development, i.e. which phase of process a defect is associated with, thus, locating and fixing the process as well as the defect can be quite straight forward.

To demonstrate the relative growth of defect types, separate growth curves can be generated. We collapsed the classes into their process associations to better observe the growth in group. This was done by dividing all the classified defect data (types) into three categories: function, interface + serialization + algorithm and assignment + checking. These categories of defects are correlated to the phases of software development process. As shown in Table 5 if a function defect is found in the system test or unit test, it points to the high-level design phase that the defect should be associated with, interface + serialization + algorithm refer to low level design and assignment + checking refers to coding phase.

4.3. Defect type arrival rate

To permit a direct comparison, only defects reported during 2007 for open source Project B and the in house project were considered. The growth curves for the collapsing of the categories of Open Source B (Figure 4) and In-house project (Figure 5) are shown. The timeline for the projects has been divided into three periods: period 0, 1 and 2. The periods were arbitrary, chosen only to better observe and analyze defect developments and do not have any process or event significance. Observe that the open source project does not suffer major function or assignment + checking defects and both of these categories are expected to stabilize soon. The interface + serialization + algorithm defects are clearly rising very rapidly in period 2 and show no sign of stabilization.

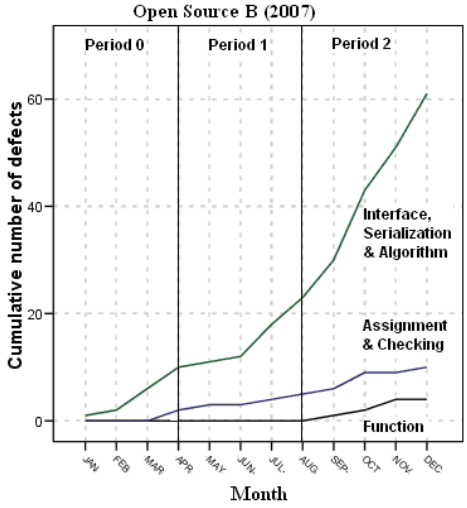

Figure 4: Project B Defect arrival by type

Overall, the open source project is functionality stable yet low level design unstable. Basically, a latest version of open source software is released in alpha and beta in which known issues have been fixed and new features have been added, thus, the software might suffer from low-level design issues. This is in contrast to the in-house project (Figure 5) where all of the defect types are stabilizing toward the end of the year.

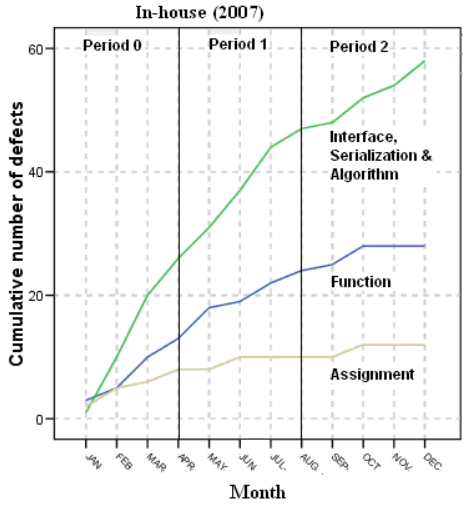

Figure 5: In house project defect arrival by type

Further classification work reveals another quality feature in open source product. Most of the defects reported for open source Project B were reported as

incorrect rather than missing (Figure 6, Figure 7, Figure 8). This is in contrast to the in house project where a significant proportion of the function and interface defects were reported as missing rather than incorrect.

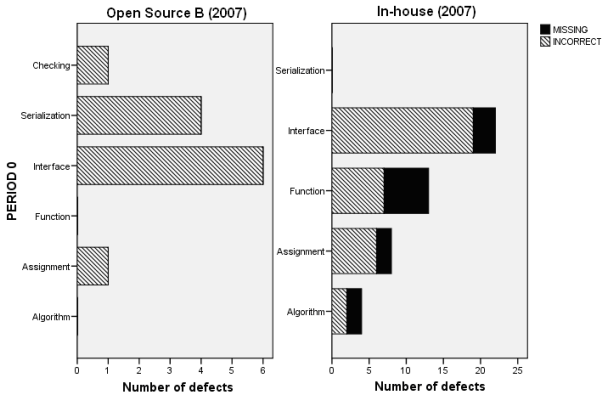

Figure 6: Frequency of defect type with qualifiers for period 0

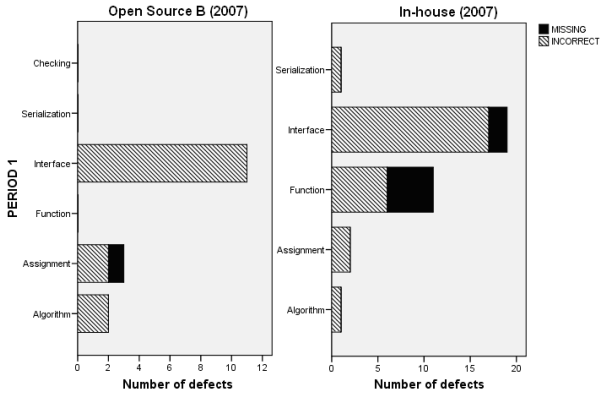

Figure 7: Frequency of defect type with qualifiers for period 1

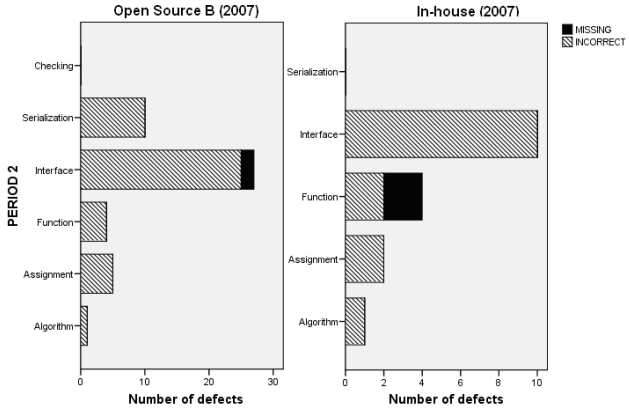

Figure 8: Frequency of defect type with qualifiers for period 2

5. Discussion

The different pattern in defect arrival of open source software seems to be a consequence of the open source software development method itself. 'Release early and release often' [15] affects the defects growth. Open source developers tend to make abrupt changes between subsequent releases due to meet new expectations such feature requests and take advantage of new technology. Despite the rapid evolution, open source does not attract defect reports of missing function and code, perhaps because of the large participation from users. This implies that there is less adherence to a formal list of requirements. Basically everyone contributes their own skills and fulfill their own requirement, thus, missing functionality is less likely in open source software.

6. Conclusion and further research

In this study we evaluate quality characteristics of open source software in an empirical way. We examine whether or not open source software has a different defect arrival rate compared to in-house developed software. Defect analysis on two open source products has shown that the common models of reliability growth, the concave and S-shaped models, the do not fit the data very well. Open source software products appear to be unstable in the area of low level design, as the observed reliability growth curves show no sign of stabilization. Interestingly open source does not suffer major missing function and code issues compared to in-house developed software.

Defect analysis can be used as a constructive reliability predictor. We will extend this investigation to more open source projects to improve the external validity of the research before reaching any firm conclusions. This will enable us to develop and evaluate reliability measures that can be given to the community to assist with decisions about adopting open source software products.

7. References

[1] (1994), 'IEEE standard classification for software anomalies', *IEEE Std 1044-1993*.

[2] Butcher, M., Munro, H. and Kratschmer, T. (2002), 'Improving software testing via ODC: Three case studies', *IBM Systems Journal*, Vol.41 no 1**,** pp. 31 - 44.

[3] Chillarege, R., Bhandari, I. S., Chaar, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K. and Wong, M. Y. (1992), 'Orthogonal defect classification-a concept for in-process measurements', *Software Engineering, IEEE Transactions on*, Vol.18**,** no 11**,** pp. 943-956.

[4] Chillarege, R. and Biyani, S. (1994), 'Identifying risk using ODC based growth models', *Proceedings Software Reliability Engineering 5th International Symposium*, pp. 282-288

[5] Chillarege, R., Kao, W.-L. and Condit, R. G. (1991), 'Defect type and its impact on the growth curve ', *Proceedings of the 13th international conference on Software engineering Austin, Texas, United States* pp. 246-255

[6] David, N. C. (2002) *Managing Software Quality with Defects*, In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment* IEEE Computer Society.

[7] Goel, A. L. (1985), 'Software Reliability Models: Assumptions, Limitations, and Applicability', *Software Engineering, IEEE Transactions on*, Vol.SE-11**,** no no.12**,** pp. 1411-1423.

[8] Goel, A. L. and Okumoto, K. (1979), 'A Time Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures', *IEEE Transactions on Reliability*, Vol.28**,** no 3**,** pp. 206-211.

[9] Grady, R. B. (1996), 'Software Failure Analysis for High-Return Process Improvement Decisions', *Hewlett-Packard Journal*, Vol.47**,** no 4.

[10] Lyu, M. R. (1996), *Handbook of Software Reliability Engineering*, Michael, R. L. (Ed), McGraw-Hill, Inc.

[11] Musa, J. D., Iannino, A. and Okumoto, K. (1987), 'Software reliability: measurement, prediction, application'**,** pp. 621.

[12] ODC ODC-5.11 (2004), *IBM research*, Available: http://www.chillarege.com/, http://www.research.ibm.com/softeng/ODC/ODC.HTM, accessed 14th Nov 2007

[13] Ohba, M. (1984), 'Software reliability analysis models', *IBM Journal of Research and Development*, Vol.28**,** no 4**,** pp. 428-443.

[14] Pankaj, J., Rajesh, M. and Todd, P. (2007), 'The When-Who-How analysis of defects for improving the quality control process', *J. Syst. Softw.*, Vol.80**,** no 4**,** pp. 584-589.

[15] Raymond, E. S. (2000), 'The Cathedral and the Bazaar', Vol.version 3.0.

[16] Wood, A. (1996), 'Predicting software reliability', *Computer*, Vol.29**,** no 11**,** pp. 69-77.