A Comparison of the Reliability Growth of Open Source and In-House Software

Sharifah Mashita Syed-Mohamad Tom McBride
Faculty of Engineering and Information Technology, University of Technology, Sydney
{sharifah, mcbride}@it.uts.edu.au

Abstract

As commercial developers have established processes to assure software quality, open source software depends largely on community usage and defect reporting to achieve some level of quality. Thus, quality of open source software may vary. We examined defects reported in two active and popular open source software projects and an in-house project. The results of this analysis indicate that the reliability growth of each is quite distinct and that the defect profile of open source software appears to be a consequence of the open source software development method itself.

Key Words- Software Reliability, Open Source Software, In-house Software, Orthogonal Defect Classification (ODC), Defect Profile

1 Introduction

Open source software is being accepted as a viable alternative to commercial software. While commercial developers have established processes to assure software quality, open source software depends largely on community usage and defect reporting to achieve some level of quality. As a consequence open source software quality may vary. Such a potential for varying levels of quality may not be uppermost in the mind of someone intending to acquire open source software and they may assume instead that all software products are of equivalent quality. However, despite some early research [21] little is known of the reliability growth or quality characteristics of open source software.

In this research we consider the question of whether or not open source software has the same reliability growth as commercial software. If it does then the same tests of product reliability can be applied. If not, then new models and new tests of product reliability growth must be developed. To investigate, we examined defects reported in both open source software development and in-house software development to determine if the defect profile differed between the two.

This paper proceeds by first describes models commonly used to characterise reliability growth of software products during their development. We then briefly describe Orthogonal Defect Classification before describing the data collection method, analysis and comparison of open source and in-house developed software defect profiles. Finally we discuss our findings before drawing some conclusions.

2 Reliability models

Software reliability models have been used for examining the degree of reliability, and thus quality, of the developed product. The fundamental approach is to model failure data to observe reliability progress and to predict future behaviour. Although there are a number of analytical models of software reliability [13], concave and S-shaped models are the two most classes models fall into [26]. The basic idea of the two models is defined as mathematical relationship that exists between time span of using (or testing) a program versus cumulative number of errors discovered. The name of the models represents their growth shapes as illustrated in Figure 1.

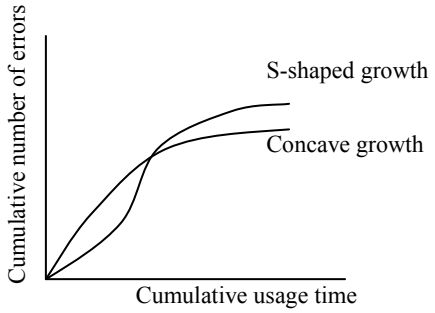

Figure 1: Concave and S-shaped reliability growth models

Goel-Okumoto nonhomogeneous Poisson model [9] and Musa model [16] are among the earliest reliability models that show concave growth curve (also called exponential). The Goel-Okumoto model describes 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 [8].

S-shaped type derives from a modification of the Goel-Okumoto model [26]. The 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.

In this research, initially we will adopt an inflection S-shaped growth (extension of the S-shaped) model as proposed by Ohba [19]. This model basically assumes that the defect discovery rate increases throughout a test period and it will have the basic exponential growth curve if the parameter r equals 1. The inflection S-shaped growth function is:

$$\mu(t) = a\left(\frac{1-e^{-bt}}{1+\psi(r)e^{-bt}}\right) \text{ where } \psi(r) = \frac{1-r}{r}, r > 0$$

$\mu(t)$ 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

$\psi(r)$ r is the inflection rate that indicates the ratio of detectable defects to the total number of defects.

We choose this model because it built on previous findings by Chillarege *et al.* [3] and Chillarege and Biyani [2]. Also, the S-shaped model is often observed in real projects [19].

3 Defect classification

3.1 Orthogonal Defect Classification

Orthogonal Defect Classification (ODC) is a classification scheme for software processes based on the attributes contained in the defect stream [1]. It provides guidance in the analysis of a classification for software defects to get a better insight into the software development process during the later parts of testing. 'Orthogonal' much like in geometry, refers to the independent characteristic captured by the defect attributes and their values that are used to classify defects. These attributes and their values are significant in exploring and understanding most software development issues.

Several studies have previously shown that defect types can be used as valuable attributes to study the behaviour of the overall reliability growth curve. The defect classification is supposedly a causal analysis mechanism which provides feedback on each individual defect, as a means to identify the nature of problems inherent in the software process. Cause-effect relationships between defect types and the consequent development efforts are presented in the ODC papers [3; 1; 2].

The ODC attributes of 'type' and 'qualifier' help to reveal the kinds of errors occurring in the software development processes. Examining relationships between the two attributes 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. The 'qualifier' attribute can take a value of either missing or incorrect. Information of the ODC types and its process associations are summarized in Table 1.

Table 1: The defect type and process associations–[1]

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	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
Build/Package/ Merge	Missing or incorrect build/package/merge	Library Tools
Documentation	Missing or incorrect documentation	Publications

4 Data collection

4.1 Defect data

To conduct this study, we first 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, to consider ethical issues in doing research which consistent with standard software engineering ethics [7]. These are two of the most successful and widely used among open source communities. Both of the chosen projects are considered stable, in production as characterized in Table 2 and Table 3.

We collected defect data from Open Source A and B from SourceForge.net bug tracking system. This

captures all of the standard defect attributes such product version, failure occurrence time, priority rating and users comment on what have been observed at the time the defect was discovered. In order to obtain a reliable defect profile, and for the purpose of this study, we restrict our attention from medium to high priority (4 – 9 in a scale where 1=lowest severity and 9=highest severity) bugs reported only. Nearly all the defects are in rank-5. Due to the reliability analysis, we excluded defect reports such duplicated defect, deleted defect, platform configuration, programming language specific problems and cosmetic design such 'look and feel' problem. As can be seen in the tables, total number of accepted defects is lower than the overall after we performed the rule. To compare the defect profile of the open source project, we focused on defect reports during 2007 for Open Source B. The total number of accepted defects is 75 out of 136.

Table 2: Open Source A details

Register date	Developer	Topic	Defects over the project lifetime	
			Overall	Accepted
Nov 2000	7	Software development	300	130

Table 3: Open Source B details

Register date	Developer	Topic	Defects over the project lifetime	
			Overall	Accepted
June 2000	8	Visualization	514	362

Defect data for a software project developed using normal commercial software processes was collected from an organization in the telecommunications industry so called 'in-house'. Defect that is found in-house refers to any fault that occurs in software that is developed in-house by the product/component development team [18]. The data is maintained in a web-based bug tracking system Again, we perform the same activities to the defect data and a summary of the in-house project is shown in Table 4.

Table 4: 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

4.2 Defect Profiles

Our primary goal is to systematically profile defects from open source and in-house projects to observe shape of the reliability growth. The classification is done manually as automatic methods of classifying defects are not usually accurate due to inherent ambiguities in language. Even so, accuracy in classifying defects may become an issue although the 'orthogonality' defect classes reduce the probability of misclassification.

We adopted the classification scheme from ODC version 5.11 [18] in our experiment. Each defect the qualifier was classified either as missing or incorrect.

To demonstrate the relative growth of defect types, separate growth curves can be generated for each of them. 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 1, 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.

5 Analysis and Findings

Our defect data collection are examined and transformed into several models for analysis using a statistical tool (SPSS). We compare defect profiles of open source and in-house source software to determine if the defect growth and decline differed between the two types of software development.

5.1 Compare defect profile

We plotted cumulative number of defects found to the growth model over the life of open source and in-house projects as presented in Figure 2. The model represents the phenomenon of software reliability growth that enables us to establish the likely pattern of open source reliability growth model.

The open source lifetimes represent the projects are in the community for over seven years by the end of 2007. Whereas the lifetime of in-house represents two final stages of software development; system testing and maintenance (field) for over two and the half years. The in-house project was in testing for two weeks before it has been officially used to provide service to customers.

Obviously, the growth curve of Open Source A and B did not exhibit the inflection S-shaped growth pattern in contrast to the in-house. In addition to the in-house

model, we have Chillarege's findings [3; 2] of reliability growth curve to be compared to our findings. Their results also demonstrate an S-shaped model, as same as the in-house.

The curve of Open Source A exhibits a reverse S-shaped curve to the in-house in which the curve at first ramp-up and levels off for sometime and begins to climb up rapidly at the half period of the calculated project life span. Meanwhile, Open Source B growth curve shows different pattern. The growth defects increases slowly in early stage and very rapidly in later stage without any sign of slowing down. Overall, both of the open source projects share a similar feature in their growth defect i.e. no sign of stabilization.

We then focus to the growth defects of Open Source B and In-house project during the recent year (2007) only (Figure 3). 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.

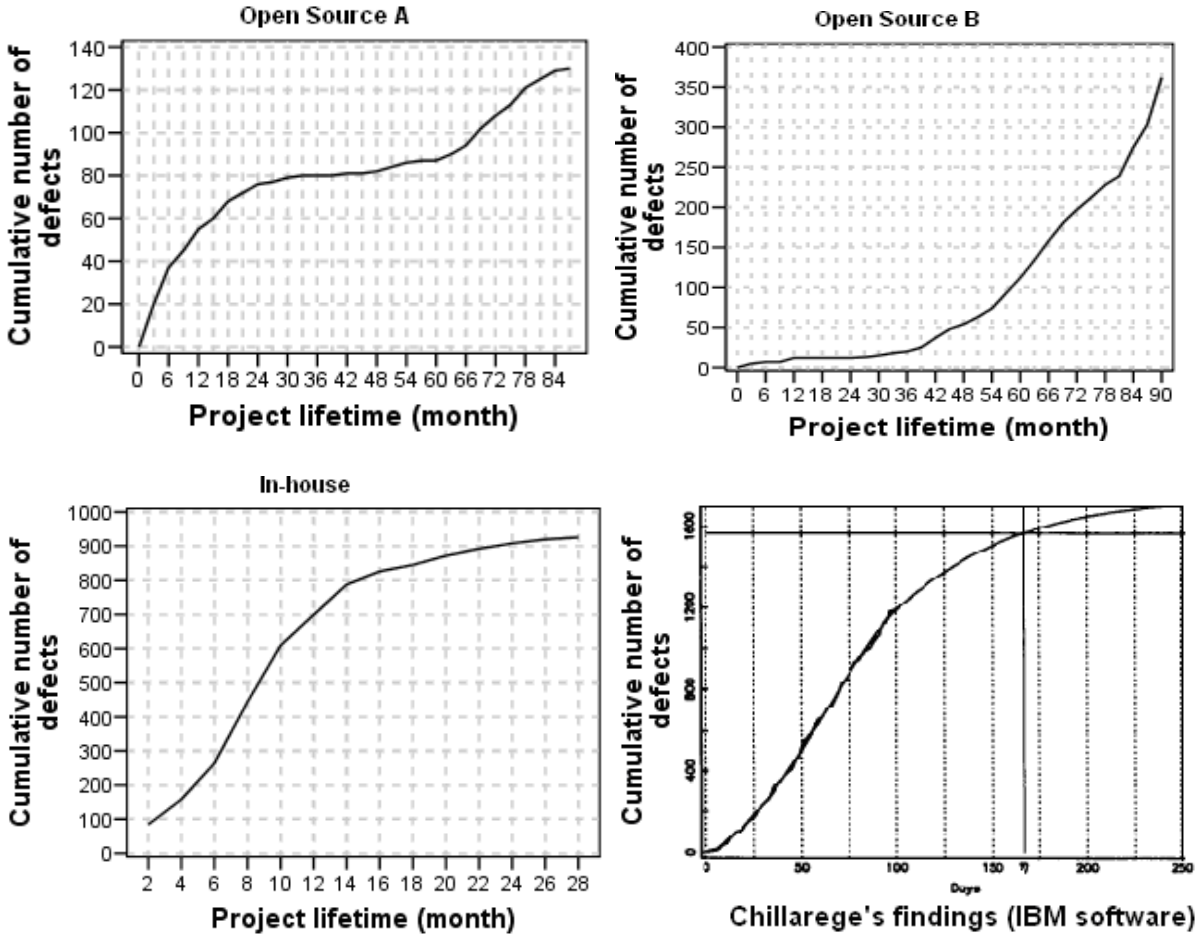

Figure 2: Cumulative defects over the life of open source and in-house project

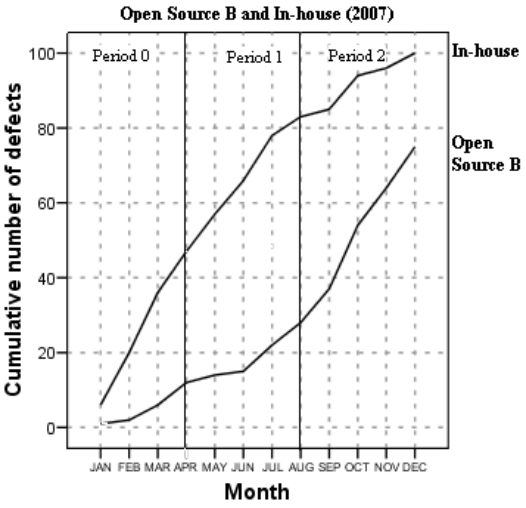

Figure 3: Overall cumulative number of development defects for open source versus in-house project over one year (2007)

Clearly, the open source growth curve illustrates a convex shape in contrast to the in-house, reflecting the fact that the two projects are at different maturity levels. This assumption is made based on the typical reliability growth models. Note the dramatic increase of the open source reported bugs in period 2 without any sign of slowing down or decrease indicates that the project (the latest version) in unstable. In contrast the number of defects for in-house increases rapidly in the first two periods and thereafter the increase noticeably slow down in period 2. The concave shape of the in-house appears to be the end of the S-curve growth model, as a signal of stabilization. The S-shaped reliability growth curve is typically caused by the continuous fixing errors during software development.

To ascertain the validity of the assumptions, we utilize the overall growth curves by combining it with the ODC defect types, as widely covered in ODC papers [3; 1; 23; 2; 21; 4]. Defect data are classified into their types and grouped according to their process associations, thus, an extension of growth model can be portrayed as shown in Figure 4. The ODC literatures establish the idea that defect type describes the nature and scope of the defect fixes, helps a development team to see what is happening in the software development. As expected, a different insight can be gleaned from this method.

Figure 4 illustrates the growth curves for the collapsing of the categories, makes the relative comparison comprehensible. Observe that the open source project does not suffer major function or assignment + checking defects and both of these categories are expected to stabilize very soon.

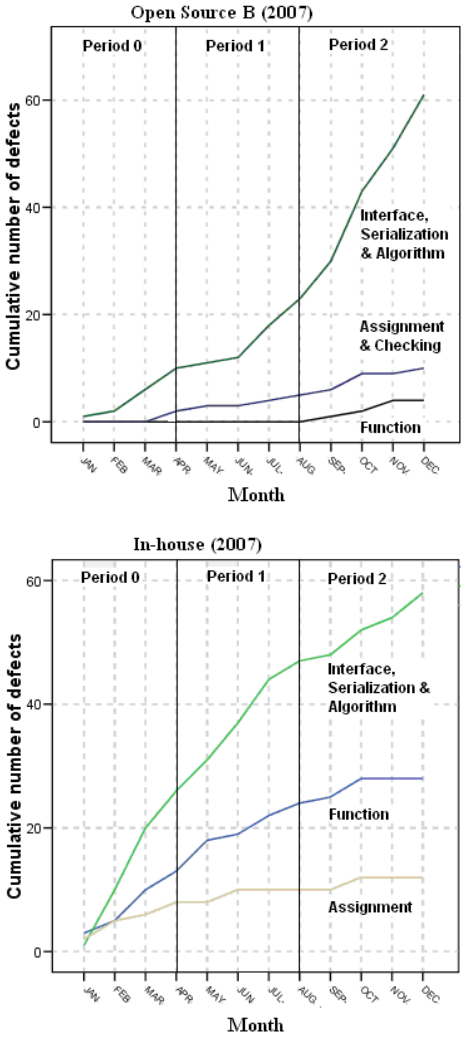

Figure 4: Growth curves for the collapsing of categories based on ODC defect type

The interface + serialization + algorithm defects are clearly rising very rapidly in period 2 and show no sign of stabilization. This means that 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 will suffer from low-level design issues.

5.2 Defect distributions

In addition to the reliability growth models, exploiting the defect type attribute of the ODC classification on defects reveals an extra defect signature. The change in defect type distribution as software moves through development can be examined by monitoring increase and decline number of defects for a specific defect type.

Figure 5 shows the change of defect type distribution of open source as the project goes through different periods during 2007. Each bar in the distribution corresponds to the fraction of defect types in the three periods. Clearly, the open source experiences significant interface defects as the project gets to the end of 2007, quite the opposite distribution to the in-house, as can be seen in Figure 6.

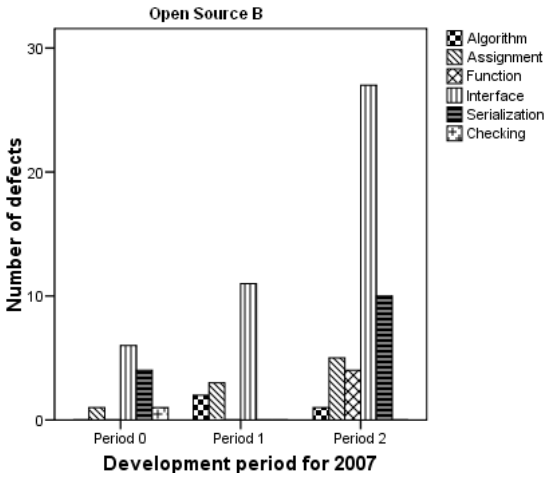

Figure 5: ODC defect type distribution of open source per year 2007

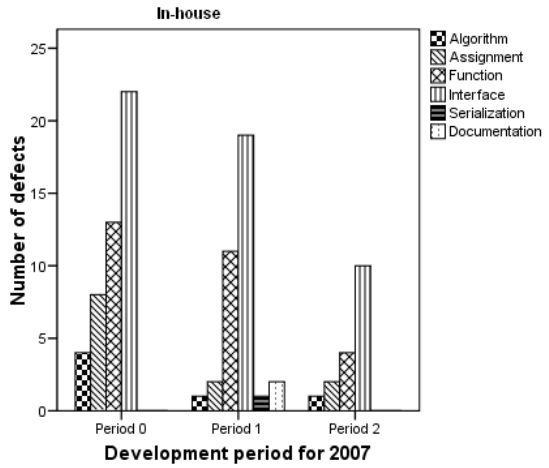

Figure 6: ODC defect type distribution of in-house per year 2007

The number of interface defects keeps decreasing along the periods, as well as other defects for the in-house project. Interface type of defect signs communication problems between separate software components/modules/devices. Despite a decline in function defects, it is clearly that the in-house suffers significant function issues compared to the open source project.

6 Discussion

What do defects and defect profiles tell us about the software product or development?

Our initial study reveals that open source software projects do have a different defect profile to in-house software products. Evidently, open source projects do not always conform to the S-shape (sigmoid) software reliability growth model. Other empirical studies in open source software show the same findings [12; 24]. Basically, the S-shape growth curve reflects to the initial learning phase, as testers become familiar with the software, followed by growth and stabilizes when there is no further defect or the residue defects become more difficult to discover.

This finding indicates that open source developers make rapid changes between subsequent releases, in which results in changes in code structure and hence bug arrival rate. Rapid development in open source software affects growth and decline of defects, as defects may be found and fixed or new functionality added quickly. Indirectly this tells us two factors that may affect the growth of defects, i.e. familiarity of open source projects and size of community. In our study, Open Source A is a software development tool for developers. In contrast, Open Source B is a visualization application for end-users, so it has larger community to post issues and request changes and thus has higher defect reports than Open Source A. In open source development, defects are located and fixed through the contribution of a large number of users and developers [17; 21; 15; 20]. Empirical studies suggest that open source projects that have more than seven developers and 100 bug reports are comparatively successful. Having several developers and bug reports indicates teamwork in a software project and at least this project achieve to attract and maintain developers over time [5; 6]. We chose our open source projects in line with these criteria.

Furthermore, it is worth mentioning that we analyzed reliability on filtered defect data, where we observed that many invalid, duplicated and cosmetic defects are being reported in open source software development. Data filtering is essential to obtain an accurate reliability analysis [10]. Our data so far indicates that open source software has a different reliability growth profile than in-house software.

We find that both open source and in-house software suffer the same low level design problems, indicated by the number of interface, serialization and algorithm defects. However, in-house project seems to show stabilization in their growth curve, while open source projects do not.

A notable signature that we discovered in our study is relatively low number of functionality issues in open source software in spite of the rapid changes between subsequent releases that occur in open source products.

These findings are interesting and point toward further research.

2. What is the cause of the difference between In-house and OS? What does this tell us about OS development?

The remarkably short of defect fix times [14] results in extremely rapid evolution of the open source software projects. We deduce that the philosophy 'release early and release often' [21] does play important role in the open source software development. Thus, it is unlikely to see the sign of stabilization in the reliability growth model of open source projects.

Even though there are rapid fixing activities and releases do occur in in-house project, interestingly yet we can see they follow the s-shape growth model. One reason this happens, we believe, is because of new feature requests by the open source community. This interesting feature draws our attention upon an issue which demands further scrutiny.

Aside from those already discussed, there are some other reasons explain the difference between in-house and open source software

1) Different strategies to manage open source quality when considering the activities of traditional software engineering. Quality assurance activities are unorganized, but extensive field testing helps improve quality [25; 27]. The software quality highly depends on the post-delivery fault reporting and correction in contrast to close source software. The quality of commercial source software is largely achieved through the systematic testing before the product release, thus post delivery fault reporting and correction play a small role [27].

2) Unmoderated changes in open source code. Without having to get permission from a principal developer [17], people can build their own work. This feature compels the code to have numerous feature requests, bug fixes and patches.

3) Co-operative development in open source. The strength of the open source community in which users as co-developers speed up debugging [17]. This can be linked to the fact of lively interest in open source software, because of the problem domain usually is well-known among technical communities. In addition, open source developers are generally end users of the product they develop, whereas in traditional software requirements engineering effort, developers and users are separate entity and developers tend not to routinely use the system they develop [22; 11].

7 Conclusion and further research

This paper presents novel and interesting findings, which are appropriate for a case study. We set out to examine whether or not open source software exhibited the same reliability growth as commercial software. Our initial research indicates that the reliability growth of each is quite distinct and that the defect profile of open source software appears to be a consequence of the open source software development method itself.

However, these are initial findings from a study of only two open source software projects and more projects need to be examined before reaching any firm conclusions.

Defect analysis has provided a useful characterization of product reliability and we will continue with this line of investigation. Our aim is to find out what method can be used to evaluate the reliability of open source products, so the community can assess the reliability of whatever open source product they want to adopt.

8 References

[1] 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.

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

[3] 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

[4] Chillarege, R. and Ram Prasad, K. (2002), 'Test and development process retrospective - a case study using ODC triggers', *Dependable Systems and Networks on International Conference*, pp. 669-678

[5] Crowston, K., Annabi, H. and Howison, J. (2003), 'Defining open source software project success', *Proceedings of the 24th International Conference on Information Systems (ICIS 2003)*, pp. 327-340

[6] Crowston, K., Annabi, H., Howison, J. and Masango, C. (2004), 'Effective work practices for software engineering: free/libre open source software development ', *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research Newport Beach, CA, USA* pp. 18-26

[7] El-Emam, K. (2001), 'Ethics and Open Source', *Empirical Softw. Engg.,* Vol.6**,** no 4**,** pp. 291-292.

[8] 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.

[9] 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.

[10] Kanoun, K., Kaniche, M. and Laprie, J.-C. (1997), 'Qualitative and Quantitative Reliability Assessment', *IEEE Softw.,* Vol.14**,** no 2**,** pp. 77-87.

[11] Koru, A. G. and Tian, J. (2004), 'Defect handling in medium and large open source projects', *Software, IEEE,* Vol.21**,** no 4**,** pp. 54-61.
[12] Li, P. L., Herbsleb, J. and Shaw, M. (2005), 'Finding predictors of field defects for open source software systems in commonly available data sources: a case study of OpenBSD', *Software Metrics, 2005. 11th IEEE International Symposium*, pp. 10 pp.
[13] Lyu, M. R. (1996), *Handbook of Software Reliability Engineering,* Michael, R. L. (Ed)*,* McGraw-Hill, Inc.
[14] McConnell, S. (1999), 'Open-source methodology: ready for prime time?' *IEEE Software July/August* Vol.16**,** no 4**,** pp. 6-11.
[15] McLaughlin, L. (2004), 'Automated bug tracking: the promise and the pitfalls', *Software, IEEE,* Vol.21**,** no 1**,** pp. 100-103.
[16] Musa, J. D. (1975), 'A theory of software reliability and its application', *IEEE transactions on software engineering,* Vol.1**,** no 3**,** pp. 312-327.
[17] O'Reilly, T. (1999), 'Lessons from open-source software development', *Commun. ACM* Vol.42**,** no 4 pp. 32-37.
[18] ODC ODC-5.11 (2004), *IBM research*, Available: http://www.research.ibm.com/softeng/ODC/ODC.HTM, accessed 14th Nov 2007
[19] Ohba, M. (1984), 'Software reliability analysis models', *IBM Journal of Research and Development,* Vol.28**,** no 4**,** pp. 428-443.
[20] Paulson, J. W., Succi, G. and Eberlein, A. (2004), 'An Empirical Study of Open-Source and Closed-Source Software Products', *IEEE Trans. Softw. Eng.,* Vol.30**,** no 4**,** pp. 246-256.
[21] Raymond, E. S. (2000) *The Cathedral and the Bazaar*, Vol. version 3.0.
[22] Scacchi, W. (2002), 'Understanding the requirements for developing open source software systems', *Software, IEE Proceedings -,* Vol.149**,** no 1**,** pp. 24-39.
[23] Sullivan, M. and Chillarege, R. (1992), 'A comparison of software defects in database management systems and operating systems', *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pp. 475-484
[24] Tamura, Y. and Yamada, S. (2006), 'A Method of User-oriented Reliability Assessment for Open Source Software and Its Applications', *Systems, Man and Cybernetics, 2006. ICSMC '06. IEEE International Conference on*, pp. 2185-2190
[25] Vixie, P. (1999) *Software Engineering* In *Open Sources: Voices from the Open Source Revolution* O'Reilly & Associates.
[26] Wood, A. (1996), 'Predicting software reliability', *Computer,* Vol.29**,** no 11**,** pp. 69-77.
[27] Yu, L. and Chen, K. (2007), 'Evaluating the Post-Delivery Fault Reporting and Correction Process in Closed-Source and Open-Source Software', *Software Quality, 2007. WoSQ'07: ICSE Workshops 2007. Fifth International Workshop on*, pp. 8-8