ISSN No. 0976-5697

Volume 2, No. 5, Sept-Oct 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Reliability of Software Development Using Open Source Technology

Razeef Mohd. and Mohsin Nazir*
Department of Information Technology
Central University of Kashmir, Srinagar, J&K (India)
m.razeef@gmail.com
*mohsin.kawoosa@yahoo.com

Abstract: The main quality attribute of a software product is the degree to which it can be relied upon to perform its intended function. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers and software from the early days of their evolution. A number of analytical models have been proposed during the past 15 years for assessing the reliability of a software system. Software reliability concerns itself with how well the software functions to meet the requirement of the user. Thus reliability incorporates all those properties that can be associated with execution of the program. For example, it includes correctness, safety and the operational aspects of reusability and user friendliness. In this paper we present an overview of the software reliability in general and reliability of open source software in particular. Furthermore in this paper various analytical models proposed to address the problem of software reliability measurement are discussed.

Keywords: Software Reliability, Software Reliability Growth Models (SRGMs), fault, failure, Weibull distribution, open source software, Rayleigh distribution, Exponential distribution, Exponentiated Weibull (EW) and Non-Homogenous Poisson Process (NHPP).

I. INTRODUCTION

The role of computer software has undergone significant change over a time span of little more than 50 years. Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer — producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

Software delivers the most important product of our time—information. Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

In this contemporary world where, to keep pace, with environment and market has become most vital part of society and business at large, the most important factor to sustain customer in competitive market in order to provide the best possible service quality, which results in improving customer satisfaction, customer retention and at the same time profitability.

Thus, the impact of the service quality concept augurs the researchers and scholars to address this issue and to investigate it further across the different service sectors. To maintain the basic theme and agenda that leads the way for development and customer centricity and connectivity is the main cause that encompasses and paves way to tread on the right track to elevate and create an atmosphere of trust among the customers and then retain them. It appears that service quality is not a new concept; however, measuring and managing service quality from the customers' point of view is still a developing and a challenging issue. It is well established that measurement of service quality is an important procedure for improving the performance of the overall service quality.

Systems with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software. For example, large next-generation aircraft will have over one million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming international Space Station will have over two million lines on-board and over ten million lines of ground support software; several major life-critical defense systems will have over five million source lines of software. While the complexity of software is inversely related to software reliability, it is directly related to other important factors in software quality, especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to software.

The amount and complexity of software produced today stagger the imagination. Software development strategies have not kept pace; however, software products fall short of meeting application objectives. Consequently controls must be developed to ensure a quality of a product. The software development life cycle includes various stages of development, and each stage has the goal of quality assurance.

The software quality contains a set of different properties. Software Reliability is one of the most important attribute of software quality, together with functionality,

usability, performance, serviceability, capability, installability, maintainability, and documentation.

II. SOFTWARE AND HARDWARE RELIABILITY

The partition between hardware and software reliability is somewhat false. We can define both the reliabilities in the same way. And then combine hardware and software component reliabilities to get system-reliability. Both depend on the environment. The source of failures in software is designed faults while the principal source in hardware has generally been physical deterioration.

However, the concepts and theories developed for software reliability could really be applied to any design activity, including hardware design. Once a software (design) defect is properly fixed, it is in general fixed for all time. Although manufacturing can affect the quality of physical components, there application process for software (design) is trivial and can be performed to very high standards of quality. Since introduction and removal of design faults occur during software development, software reliability may be expected to vary during this period. The "design reliability" concept has not been applied to hardware to any extent. It was possible to keep hardly generally less complex logically then software.

A. Software Reliability:

Software Reliability is a subfield of software engineering in which practitioners are concerned with measuring and managing software quality. This valuable discipline has inherited much of its theory from hardware reliability and has gained mixed acceptance from the software community. Software Reliability has been regarded as one of the important quality attributes because low reliable software systems have high possibilities of causing serious problems such as the loss of human life, catastrophic mission failures, and the waste of valuable resource investments. Software reliability can be viewed as a powerful measure of quantifying software failures and is defined as the probability of failure-free software operation for a specified period of time in a specified environment.

Therefore, in order to achieve a desired level of quality, the reliability of a software system must be high. The fault-detection and fault correction are critical processes in attaining good software quality. During the software detection process, testing cases are run and ultimately failures are detected. After detection, the debugging team should analyze the failure, locate the fault and fix the fault. That is, the fault correction process affects the reliability of a software product significantly and we should pay more attention to it.

Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment [12, 18, 20, 21]. It is evident from the definition that there are four key elements associated with the reliability namely element of probability, function of the product, environmental conditions, and time. Software Reliability is not a function of time - although researchers have come up with models relating the two. The software reliability also can be expressed as the intensity of the defects- number of defects in the unit time. The relationship between the intensity of defects and reliability depend upon the model used for evaluation. It is difficult to establish the fact that a model is better than others.

B. Hardware Reliability:

Hardware Reliability is concerned with the random occurrences of undesirable events, or failures, during the life cycle of a physical system. Since a failure phenomenon can only be described in probability terms, the definition of reliability depends heavily on probability concepts. The reliability of a system is defined as the probability that the system will adequately perform its intended function for a specified interval of time under stated environment conditions. Reliability evaluation using probability methods provides a quantitative measure of system performance. Hence it allows comparison between systems or provides a logical basis for reliability improvement in a system.

A straight approach of the reliability supposes its orientation rather to the user than to the software development process. This approach derives from the users' point of view, which determinates an easier understanding of the reliability by the clients. Also, refers more at the execution than to the design, which makes it more dynamic than statically. One advantage is the fact that reliability is being calculated for the both components: hardware and software. We can also calculate the reliability to the entire system.

The software reliability management is defined as the software reliability improvement process that accentuate on warning, detecting and elimination of the software errors and on using the metrics in order to maximize the reliability by its project compulsions-resources, the cost and the performances. For reliability maximization we must do:

a. Error Prevention.
b. to detect and eliminate errors;
c. Measurements for increasing reliability, for define and for using specific metrics to sustain first two activities.

Reliability determination implies sustained efforts which takes into account the medium time between errors and the medium time until appears first error. This date represents in-puts for the successful models which had been built and which realize previsions of the error rate and of the reliability.

Reliability is the quality product and quality can be measured. In order to be effective, the measure must be a part of the management activity. The measures help in order to achieve the fundamental objectives of the management, concerning prevision, progress, and processes improvement.

III. MODELING

A model is a simple representation of the system or real process comportment or structure. The goal of the modeling process is to reproduce the fundamental relationships in order to realize their clear understanding.

In order to model software reliability we must first consider the principal factors that affect it: fault introduction, fault removal, and the environment. Fault introduction depends primarily on the characteristics of developed code (code created or modified for the application) and development process characteristics. The most significant code characteristic is size. Development process characteristics include Software Engineering technologies and tools used and level of experience of

personnel. Code can be developed to add features or remove faults. Fault removal depends on time, operational profile (set of run types that the program can execute along with the probabilities with which they occur) and the quality of repair activity. The environment directly depends on operational profile. Since some of the foregoing factors are probabilistic in nature and operate over time, software reliability models are generally formulated in terms of random process. The models are distinguished from each other in terms of probability distribution of failure times or number of failures experienced and by the nature of the variations of the random process with time. A software reliability models specifies the general form of the dependence of the failure process.

A Software Reliability Growth Model is one of the fundamental techniques to assess software reliability quantitatively. The Software Reliability Growth Model required having a good performance in terms of goodness-of-fit, predictability, and so forth. In order to estimate as well as to predict the reliability of software systems, failure data need to be properly measured by various means during software development and operational phases. Any software required to operate reliably must still undergo extensive testing and debugging. This can be a costly and time consuming process, and managers require accurate information about how software reliability grows as a result of this process in order to effectively manage their budgets and projects. The effects of this process, by which it is hoped software is made more reliable, can be modeled through the use of Software Reliability Growth Models, hereafter referred to as SRGMs. Research efforts in software reliability engineering have been conducted over the past three decades and many software reliability growth models (SRGMs) have been proposed. SRGMs can estimate the number of initial faults, the software reliability, the failure intensity, the mean time-interval between failures, etc.

Ideally, these models provide a means of characterizing the development process and enable software reliability practitioners to make predictions about the expected future reliability of software under development. Such techniques allow managers to accurately allocate time, money, and human resources to a project, and assess when a piece of software has reached a point where it can be released with some level of confidence in its reliability. Unfortunately, these models are often inaccurate. A comparative study of Software Reliability Growth Models [41] allows to determine with models is suited well and under what conditions.

Many SRGMs based on NHPP which incorporates the testing–effort functions (TEF) have been proposed by many authors (Yamada et al., 1984; 1986; 1993; Yamada and Ohtera, 1990; Huang et al., 2007; Kuo et al., 2001; Bokhari and Ahmad, 2006; Quadri et al., 2006). Recently, Bokhari and Ahmad (2007) and Ahmad et al. (2008; 2010) also proposed a new SRGM with the Exponentiated Weibull (EW) testing-effort functions to predict the behavior of failure and fault of software.

A. Characteristics of a Software Reliability Model:

A good model presents following properties: [23]

a. It should provide good prediction of future behavior.
b. It should compute useful quantities.
c. It should be simple.
d. It should be widely applicable.
e. It should be based on sound assumptions.

The software reliability modeling is a functional representation of the debated system and the better model offers a viable mechanism for reliability estimation.

B. Analytical Models:

A number of analytical models have been proposed to address the problem of software reliability measurement. These approaches are based mainly on the failure history of software and can be classified according to the nature of the failure process studied as indicated below:

a. Times between Failures Models :

In this class of models, the process under study is the time between failures. The most common approach is to assume that the time between, say, the $(i-1)$st and the *ith* failures, follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. Estimates of the parameters are obtained from the observed values of times between failures and estimates of software reliability, mean time to next failure, etc., are then obtained from the fitted model. Another approach is to treat the failure times as realizations of a stochastic process and use an appropriate time-series model to describe the underlying failure process.

b. Failure Count Models:

The interest of this class of models is in the number of faults or failures in specified time intervals rather than in times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. Parameters of the failure rate can be estimated from the observed values of failure counts or from failure times. Estimates of software reliability mean time to next failure, etc., can again be obtained from the relevant equations.

c. Fault Seeding Models:

The basic approach in this class of models is to "seed" a known number of faults in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed numbers of seeded and indigenous faults are counted. From these, an estimate of the fault content of the program prior to seeding is obtained and used to assess software reliability and other relevant measures.

d. Input Domain Based Models:

The basic approach taken here is to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program. Because of the difficulty in obtaining this distribution, the input domain is partitioned into a set of equivalence classes, each of which is usually associated with a program path. An estimate of program reliability is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain

IV. RELIABILITY OF OPEN SOURCE TECHNOLOGY

Open source technology has gained a significant amount of mind share and has been the subject of much debate. Often promoted as being better than proprietary software

(from an ethical and social point of view), and criticized as being unrealistic or too idealistic. The concept itself is based on the philosophy of free software, which advocates freely available source code as a fundamental right. However, open source extends this ideology slightly to present a more commercial approach that includes both a business model and development methodology.

According to the "Open Source Initiative" software is considered "open source" if its distribution terms adhere to the following:

a. ***Free Redistribution –*** Copies of the software can be made at no cost.

b. ***Source Code –*** The source code must be distributed with the original work, as well as all derived works.

c. ***Derived Works –*** Modifications are allowed, however it is not required that the derived work be subject to the same license terms as the original work.

d. ***Integrity of the Author's Source Code –*** Modifications to the original work may be restricted only if the distribution of patches is allowed. Derived works may be required to carry a different name or version number from the original software.

e. ***No Discrimination Against Persons or Groups*** Discrimination against any person or group of persons is not allowed.

f. ***No Discrimination Against Fields of Endeavor***

g. ***Restrictions Preventing use of the Software by a Certain Business or Area of Research are not Allowed.***

h. ***Distribution of License –*** Any terms should apply automatically without written authorization.

i. ***License Must Not Be Specific to a Product –*** Rights attached to a program must not depend on that program being part of a specific software distribution.

j. ***License Must Not Contaminate Other Software –*** Restrictions on other software distributed with the licensed software are not allowed.

In general Open Source Software (OSS) refers to any software whose source code is freely available for distribution. The success and benefits of OSS can be attributed to many factors such as code modification by any party as the needs arise, promotion of software reliability and quality due to peer review and collaboration among many volunteer programmers from different organizations, and the fact that the knowledge-base is not bound to a particular organization, which allows for faster development and the likelihood of the software to be available for different platforms. Eric Raymond states that "with enough eye balls, all bugs are shallow", which suggests that there exist a positive relationship between the number of people involved, bug numbers, and software quality. Some examples of successful OSS products are Apache HTTP server and the Mozilla Firefox internet browser.

An open source program typically consists of multiple modules [8]. Attributes of the reliability models have been usually defined with respect to time with four general ways to characterize [29, 24] reliability, time of failure, time interval between failures, cumulative number of faults up to a period of time and failure found in a time interval. The present methodology involves defining an equation for the pattern of failure based on the available bug arrival rate and developing a generalized model for the reliability of the software. The following are the assumptions involved in the analysis.

a. The software analyzed is an open source.
b. As the open source software is made up of a very large community the environmental changes are not considered.
c. The total number of packages at the beginning of the analysis is assumed to remain constant and is taken as the initial population.
d. The failures of various packages are assumed to be independent of each other.
e. The model is developed for evaluation of the software reliability at the developmental stage and the packages that fail during this period are not further considered. It is further assumed that by the end of developmental stage the bug associated with the failed packages would be eliminated and will be stable further.
f. The reliability of the software is inversely proportional to the number of bugs reported at any point of time.
g. The beginning of the time period after which the bug arrival or failure rate remains constant marks the culmination of the developmental stage and the software will be stabilize.

V. ERROR ESTIMATION FOR OPEN SOURCE SOFTWARE

Many models have been proposed to assess whether a software-testing objective has been met to determine when to stop testing. These models are based on various sets of assumptions about the software and its execution environment. Software reliability growth models (SRGM`s) use data about the times that failures occur to estimate the number of remaining failures in a system [26]. Generally, SRGM`s estimate the number of failures in a system.

However, existing empirical evidence shows that SRGM`s can also be applied on error data to estimate the number of defects in a system [26], [10]. For the software practitioner, the assumptions or conditions for the various SRGM`s are an open problem, because they are often violated in one way or another. SRGM`s are usually robust, despite these assumption violations [26] [10].There are empirical selection method that aids in choosing the appropriate SRGM model when assumptions are not met [10].

The approach for the project described includes three phases. The first phase involves selecting, downloading and installing defect-tracking tools that are freeware or open source. Several tools are installed on both Windows and Linux platforms. The second phase involves documenting the format of the data collected by the tools and implementing data extraction and conversion routines to transform data from the various defect tracking software tools into a consistent delimited format for a comprehensive defect estimation tool. The third phase involves analyzing the defect data for the tools to determine if any defect estimation method can be applied.

Seven defect-tracking tools were installed on either Windows or Linux platforms. Those tools include Buggit, Bugtrack, Bugs Online on the Windows platform and Bugzilla, Roundup, Mantis and Incident Management Systems (IMS) on the Linux side. Table 1 provides the details about each tool, such as the supporting software

needed, effort to install, and the usability of the tool in terms of how easy it is to learn and use. Roundup was immediately rejected, because the data could not be easily exported from the tool.

Table I.Defect tracking tools analyzed [30]

	Buggit	**Bugzilla**	**Bugtrack**	**Roundup**	**Mantis**	**IMS**	**Bugs Online**
Platform	Windows 2000/XP	Linux (Fedora)	Windows 2000	Linux (Fedora)	Linux (Fedora)	Linux (Fedora)	Windows
Database	MS access	MySQL	SQL Server 2000	MySQL	MySQL	MySQL	MS access
Server (if any)		Apache	IIS 5.0	Apache	Apache	Apache	IIS 5.0
Other SW		Perl modules		Python	php	Php/Zend optimizer	
Effort to install	Low	High	High	Med	High	Med	Med
Effort to learn &use	Low	Med	Low	High	Med	Med	Med
Ability to export data	Export from MS Accesss	Export using phpMYAdmin	Export from SQL Server 2000	Cannot export data	Export using phpMYAdmin	Export using phpMYAdmin	Export from MS Accesss

In table 1 are detailed information on the kinds of data reported for each defect by the tools. Analysis of the database formats of these defect tracking tools lead to several observations. All of the tools have an ID or a name for each bug in their databases, which is necessary to identify each defect uniquely. Four tools have data about the components in which a defect exists. This is useful in estimating fault-content by modules or components. The component may be a subsystem, a module, or a file. Almost all defect-tracking tools analyzed have data that give the date a defect report is created or added. For applying SRGM`s, the add date or the open date of a defect is necessary. Almost all tools include severity levels in their data. Metrics by severity are very helpful in improving software reliability growth models (SRGM`s) [26], [10].

VI. RELIABILITY MODELS FOR OPEN SOURCE SOFTWARE

Concern about software reliability has been around for a long time and as open source is a relatively novel software development approach differing significantly from proprietary software waterfall model, we do not yet have any mature or stable technique to assess open source software reliability. Weibull distribution and Reliability Growth Model are possible ways to establish the reliability model.

A. The Weibull Family Models:

Weibull distribution family is perhaps the most widely used lifetime distribution model [19]. Its simplest form, the 2-parameter Weibull distribution, has long been used to model reliability pattern due to its ability in describing failure modes like initial, random and wear-out [31]. Data from large commercial software suggests two special forms of Weibull distribution: Rayleigh distribution and exponential distribution have been applied in software reliability models [17].

The 2-parameter Weibull distribution has a probability distribution function of the form:

$$f(t) = \frac{(\beta t)^{(\beta - 1)}}{\alpha^{\beta}} e^{-(t/\alpha)^{\beta}} \qquad (1)$$

Where t represents time; $\alpha = 1/\lambda$ represents the scale parameter of the distribution and β represents the shape parameter of the distribution. The Weibull probability density function is monotone decreasing if $\beta <= 1$ and becomes bell shaped when $\beta > 1$. The larger the β value the steeper the bell shape. Its special case Rayleigh distribution has $\beta = 2$; while exponential distribution has $\beta = 1$. Figure 1 shows several Weibull probability density curves with varying values for the shape parameter β.

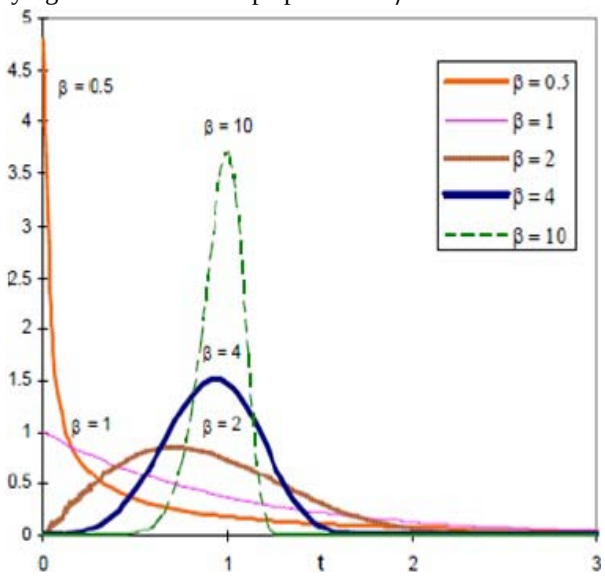

Figure: 1

In software quality engineering, large body of empirical data supports the finding that software projects follow a life cycle pattern described by Rayleigh curve. This is considered as a desirable pattern since the bug arrival rate stabilizes at a very low level. In closed source software, the stabilizing behavior is usually an indicator of ending test effort and releasing the software to the field. The development cycle, from quality perspective, is divided into six phases, high-level design inspection, low-level design inspection, code inspection, unit test, component test and system test. The bug arrivals usually peak at the code inspection phase and are rather stabilized in the system test phase [13].

a. Data Collection Phase:

In the data collection stage, users first identified a few target open source projects according to product size. The main selection criteria include project duration and activity. New projects with less than one-year history usually will not

be able to contribute enough useful information. One typical phenomenon associated with many open source projects is that they might get on and off during the project period. To rank projects according to activities, SourceForge.net devised an activity measure. It takes number of downloads, number of times mentioned in the forum and other measures into consideration and combine them to form an activity index. We picked our target projects following SourceForge's "Most active" rank list as of August, 2004.

The projects identified are listed in table 2 along with their prominent features. Actual names of the projects were not revealed to follow standard software engineering ethics [5].

Users mainly collected information regarding newly opened bugs per month. SourceForge provides simple statistics on monthly bug reported. However, these statistics show all bugs reported regardless whether they are valid or not. Each month, there are a few bugs reported and then deleted by the project owner or core developer members. They are considered as invalid bugs for various reasons; we need to exclude those bugs for accuracy.

Besides, those statistics are collections of all software products under one project title, it is more sensible to differentiate among different products. Fortunately, SourceForge keeps detailed description for each reported bug, its status and the component it belongs to. We used the query function provided to isolate bugs from a certain component as well as each month's deleted report and rule them out from our data. Hence, the original data we got for monthly opened bugs are slightly different from those reported in SourceForge statistics.

Table II.Target Open Source Projects

Project Title	Starting Time	Developer number
PrA	2003-11-08	14
PrB	2007-05-19	9
PrC	2007-02-11	40
PrD	2004-08-21	1
PrE	2005-01-15	8
PrF	2005-01-12	115
PrG	2004-02-20	40
PrH	2004-06-23	2

b. Data Analysis Phase:

The data show similar trend across all projects except for a few projects, which are still in their infancies. The monthly bug arrival rate goes slowly upwards along until it reaches a peak; it then starts to decrease, stabilizes at a rather low level. The trend is consistent in PrA, PrE, PrF, PrD, PrG and PrH projects. Two other projects, PrB and PrC are registered for short times - around 1 year.

Since the monthly bug rate keep increasing, it is hard to tell at this stage whether they are going to reach the peak and then decrease to a stabilizing state. The monthly bug data reveal a clear pattern and there are families of models in the analysis of failure process data that fit this general distribution. However, users will concentrate on the Weibull distribution, which has long been demonstrated its appropriateness in reliability/failure time analysis [23].

VII. CONCLUSION

In this paper we discuss software reliability in general and modeling of software reliability using software reliability growth models in particular. We tried to give a general overview of the reliability of open source software and also gave a comparison of reliability of some open source software. We then discussed various various analytical models proposed to address the problem of software reliability measurement. In this paper we have also discussed reliability models for open source software. Distinctive feature of this research is that we do not add any new models to the already large collection of SRGMs.

VIII. REFERENCES

[1]. Adalberto Nobiato Crespo, Alberto Pasquini ,"Applying Code Coverage Approach to an Infinite Failure Software Reliabilit Model" , 2009 XXIII Brazilian Symposium on Software Engineering, 2009 IEEE.

[2]. Alex Bishop. Major roadmap update centers around phoenix, thunderbird; 1.4 branch to replace 1.0; changes planned for module ownership model. MozillaZine (online), April 2 2003. http://www.mozillazine.org/articles/article3042.html

[3]. Brendan Eich and Mitchell Baker. Mozilla .super-review.Web page, cited September 6, 2006., Mozilla Foundation, June 2000. http://www.mozilla.org/hacking/reviewers.html

[4]. Christian Robottom Reis and RenataPontin de Mattos Fortes. An overview of the software engineering process in the mozilla project.In *Proceedings of the Open Source Software Development Workshop*, Newcastle upon Tyne, UK, February 2002.

[5]. E. Emam, Ethics and Open source. Empirical Software Engineering, No. 4-6, 2001, pp.291-292.[35] http://sourceforge.net/, last visited in 2006.

[6]. E.S. Raymond, "The cathedral and the bazaar: musings on linux and open source by an accidental revolutionary, 2nd Ed., O'Reilly, 2001.

[7]. Fenghong Zou , Joseph Davis "Analysing and Modeling Open Source Software Bug Report Data', 19th Australian Confeence on Software Engineering.,2008 IEEE.

[8]. Fenghong Zou , Joseph Davis " A Model of Bug Dynamics for Open Source ", The second International Conference on Secure System Integration and Reliability Improvement., 2008 IEEE.

[9]. gboone. Open new window in background (tabbed browsing).http://groups.google.com/group/netscape.public.mozilla.wishlist/tree/br%owse_frm/thread/ef62c3307e2a7a32/4ec071eae14082ff?rnum=1&hl=en&_done=%2Fgroup%%2Fnetscape.public.mozilla.wishlist%2Fbrowse_frm%2Fthread%2Fef62c3307e2a7a32%2F%4ec071eae14082ff%3Ftvc%3D1%26hl%3Den%26#doc_4b33ef52c30564cf.

[10]. G. N. Rodrigues, D. Rosenblum and W. Emmerich, "A model driven approach for software systems reliability," Proceedings of the 26th International Conference on Software Engineering, 2004.

[11]. H.J. van Rantwijk. Multizilla's home page.http://multizilla.mozdev.org, February 24 2006. Home page for the MultiZilla project, cited September 6, 2006.

[12]. H. Pham, Software Reliability. Springer-Verlag, 2000.

[13]. H. S. Kan, Metrics and models in software quality engineering, 2nd edition, Addison-Wesley, 2003.

[14]. http://www.opensource.org/licenses/alphabetical

[15]. http://www.damicon.com/resources/opensource.html

[16]. Hudson, A, "Program Error as a British and Death Process", Technical Report SP- 3011, Santa Monica,Cal,:Systems development Corporation, 1967.

[17]. I. Samoladas, I. Stamelos, L. Angelis and A. Oikonomou, "Open source software development should strive for even greater code maintainability," Communications of the ACM, Vol. 47, No. 10, October 2004.

[18]. J.D. Musa and K. Okumoto, "A logarithmic poisson execution time model for software reliability measurement", 7th Int'l Conference on Software Engineering (ICSE), 1984, pp. 230-238.

[19]. J. F. Lawless, Statistical Models and Methods for Lifetime Data, 2nd edition, New York, 2003.

[20]. Kan H.S. Metrics and models in software quality engineering, 2nd edition, Addison-Wesley(2003)

[21]. Lars M.Karg, Michael Grottke, Arne Beckhausa . Conformance Quality and Failure Costs in the Software Industry: An Empirical Analysis of Open Source Software. 2009 IEEE.

[22]. M. P. Cristescu, Modele de evaluare a fiabilităţiisistemelorde programe, PhD thesis, Bucuresti, 2003.

[23]. M. P. Cristescu, "Methods for software reliability determination,", The 13-th International Conference „The Knowledge Based Organization" - Computer Science, Modeling and Simulation, e-Learning Technologies and Solutions for the Engineering Domain – Conference Proceedings 11, Sibiu, 22-25 November 2007, pp. 66-72, Land Forces Academy "NicolaeBălcescu" Sibiu.

[24]. Musa, J.D., Iannino, A. and Okumoto, K. (1987),"Software Reliability: Measurement, Prediction, Application', pp. 621.

[25]. Rachel Rosmarin. Mozilla _refox gaining ground on microsoft IE. Forbes.com, August 12, 2006.

[26]. R. Lincke and W. Lowe, Compendium of SW Quality Standards and Metrics, available at: http://www.arisa.se/compendium/, 2005.

[27]. Peter Bojanic. The joy of xul. Web page, cited september 6, 2006., Mozilla Foundation,June 2006.http://developer.mozilla.org/en/docs/The_Joy_of_XUL

[28]. Quadri, S.M.K., Mohd Razeef,"A Comparative Overview of Software Reliability Growth Models" International Journal of Advanced Research in Computer Science, Volume 2, No. 1, Jan-Feb 2011, pp. 99-104.

[29]. Sharifah Mashita Syed-Mohamad, Tom McBride, 'Reliability Growth of Open Source Software using Defect Analysis', 2008 International conference on Computer Science and Software Engineering, 2008 IEEE.

[30]. Steve Hamm. A _refox in IE's henhouse. Business Week, September 17 2004.

[31]. S. Yamada, J. Hishitani and S. Osaki, "Software-Reliability Growth with a Weibull Test- Effort: A model & application," IEEE Transactions on Reliability, Vol. 42, No. 1, March 1993.

[32]. unknown. A guide to mozilla 1.0.http://www.mozilla.org/start/1.0/guide/, 2002.Web page describing release 1.0 of Mozilla.

[33]. VladimirNeyman. Open new window in background. http://groups.google.com/group/netscape.public.mozilla.wishlist/tree/br%owse_frm/thread/ef62c3307e2a7a32/4ec071eae14082ff?rnum=1&hl=en&_done=%2Fgroup%%2Fnetscape.public.mozilla.wishlist%2Fbrowse_frm%2Fthread%2Fef62c3307e2a7a32%2F%4ec071eae14082ff%3Ftvc%3D1%26hl%3Den%26#doc_4ec071eae14082ff, June 23 1999. Message posted to Netscape.public.mozilla.wishlist mailing list.

[34]. Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings . Software*, 149(1):24.39, February 2002

[35]. Wonko The Sane. none.http://gnomesupport.org/forums/viewtopic.php?t=3603&highlight=&sid=c5f4%e5ae34765db22bac227d7f8b17cb,September 22 2003. Posting to the Gnome desktop user support forum.

[36]. Ying ZHOU, Joseph Davis. Open Source Software reliability model: an empirical approach, ACM 2005.