

An Experimental Analysis of Open Source Software Reliability*

Cobra Rahmani, Harvey Siy, Azad Azadmanesh
College of Information Science & Technology
University of Nebraska-Omaha
Omaha, U.S.
E-mail: (crahmani, hsiy, azad) @unomaha.edu

Abstract— Arrival failure times for five popular open source projects are analyzed and compared. The projects considered are *Eclipse*, *Apache HTTP Server 2*, *Firefox*, *MPlayer OS X*, and *ClamWin Free Antivirus*. The black-box reliability analysis of the projects is a 3-step approach that consists of bug-gathering, bug-filtering, and bug-analysis. Additionally, the reliabilities of multiple release versions of *Eclipse* are compared. It is projected that the future release of the projects under the same operational environment to follow similar reliability patterns. The results exhibit that *Weibull* distribution is a powerful analytical tool to be used in reliability growth modeling of such products.

Keywords- *Open source software (OSS); Software reliability model; Software architecture; Reliability growth model; Weibull distribution*

I. INTRODUCTION

Open Source Software (OSS) in general 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 in [17] states that “with enough eye balls, all bugs are shallow”, which suggests that there exists 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.

As software products have become increasingly complex, software reliability is a growing concern, which is defined as the *probability of failure free operation of a computer program in a specified environment for a specified period of time* [14], [16]. Reliability growth modeling has been one approach to address software reliability concern, which dates back to early 1970’s [3], [7], [19]. Reliability modeling enables the measurement and prediction of software behaviors such as Mean Time To

Failure (MTTF), future product reliability, testing period, and planning for product release time.

Software reliability growth models generally fall into two major classes: *time between failures* and *fault count* models. The main input parameter to the “time between failures” models is the intervals of successful operations. As the failures occur and fixed, it is expected that these intervals to increase. The pattern of these intervals is reflected by a probability distribution model whose parameters are estimated from simulation, testing, and operation profiles. Some examples that belong to this class of reliability modeling are Jelinski-Moranda and Littlewood models [7], [10].

The “fault count” class labeled as such because the input parameter of study is the number of faults in a specified period of time rather than the times between failures. Normally the failure rate, defined as the number of failures per hour, is used as the parameter of a Probability Distribution Function (PDF). Like the first class, as the fault counts drop, the reliability is expected to increase [8], [9], [16]. Examples of this class are Goel-Okumoto and Musa-Okumoto models [4], [14].

This study is concerned with the “fault count” of five popular OSS products: *Eclipse*, *Apache 2*, *Firefox*, *MPlayer OS X*, and *ClamWin Free Antivirus*. These projects are selected because of their high number of downloads, length of project operation, and sufficient number of bug reports. MPlayer OS X and ClamWin Free Antivirus are two projects, which can be found in sourceforge.net [20]. MPlayer OS X, launched in 2002, is a project based on MPlayer, which is a movie player for Linux with more than six million downloads. ClamWin Free Antivirus was launched in 2004 that has had more than 19 million downloads. Eclipse, Apache 2, and Firefox are the other three OSS projects, which use Bugzilla [1] as their bug-repository system. Bugzilla is a popular bug-repository system that allows users to send information about a detected bug such as bug description, severity, and reporting time.

Additionally, these projects are well-known and well-established, which have been in operation for more than four years. For software projects that have not been in operation long enough, the failure data collected may not be sufficient to provide a decent picture of software quality, which may lead to anomalous reliability estimates [12], [21]. The following table reflects on the number of years that these projects have been in operation.

* This research is funded in part by Department of Defense (DoD)/Air Force Office of Scientific Research (AFOSR), NSF Award Number FA9550-07-1-0499, under the title “High Assurance Software”.

TABLE I. RELEASE DATES OF THE OSS PROJECTS

Project Name	First official release date
Firefox	11/09/2004
Eclipse	11/07/2001
Apache 2	03/10/2000
ClamWin Free Antivirus	03/26/2004
MPlayer	09/20/2002

The Weibull distribution function is employed for the reliability analysis of the aforementioned OSS projects. Weibull distribution is widely utilized in lifetime data analysis because of its flexibility in modeling different phases of bathtub reliability, i.e. decreasing, constant, and increasing failure rates. The function has been particularly valuable for situations for which the data samples are relatively small, such as in maintenance studies [13].

The rest of the paper is organized as follows. Section 2 provides some definitions and background information. Section 3 concentrates on failure data analysis and the reliability modeling process. Section 4 concludes the paper with a summary.

II. BACKGROUND

White-box and *black-box* models are two approaches for predication of software reliability. The white-box models attempt to measure the quality of a software system based on its structure that is normally architected during the specification and design of the product. Relationship of software components and their correlation are thus the focus for software reliability measurement [2], [5], [22], [23]. In the black-box approach, the entire software system is treated as a single entity, thus ignoring software structures and components interdependencies. These models tend to measure and predict software quality in the later phases of software development, such as testing or operation phase. The models rely on the testing data collected over an observed time period. Some popular examples are: Yamada S-Shape, Littlewood-Verrall, Jelinski-Moranda, Musa-Okumoto, and Goel-Okumoto [4], [10], [11], [14], [24]. This study is concentrated on the black-box reliability approach to measure and compare the reliability of the selected OSS projects.

A fault or bug is a defect in software that has the potential to cause the software to fail. An error is a measured value or condition that deviates from the correct state of software during operation. A failure is the inability of the software product to deliver one of its services. Therefore, a fault is the cause for an error, and software that has a bug may not encounter an error that leads to a failure. Failure behavior can be reflected in various ways such as Probability Density Function (PDF) and Cumulative Distribution Function (CDF). PDF, denoted as $f(t)$, shows the relative concentration of data samples at different points of measurement scale, such that the area under the graph is unity. CDF, denoted as $F(t)$, is another way to present the pattern of observed data under study. CDF describes the

probability distribution of the random variable, T , i.e. the probability that the random variable T assumes a value less than or equal to the specified value t . In other words,

$$F(t) = P(T \leq t) = \int_{-\infty}^t f(x) dx \Rightarrow f(t) = F'(t)$$

Therefore, $f(t)$ is the rate of change of $F(t)$. If the random variable T denotes the failure time, $F(t)$, or unreliability, is the probability that the system will fail by time t . Consequently, the reliability $R(t)$ is the probability that the system will not fail by time t [16], i.e.:

$$R(t) = P(T > t) = \int_t^{\infty} f(x) dx \Rightarrow R(t) = 1 - F(t)$$

This paper will show that the relative frequency of failure times follows the Weibull PDF:

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

where α is the scale parameter and β represents the shape parameter of the distribution. The effect of the scale parameter is to squeeze or stretch the distribution. The Weibull PDF is monotone decreasing if $\beta \leq 1$. The smaller β , the more rapid the decrease is. It becomes bell shaped when $\beta > 1$, and the larger β , the steeper the bell shape will be. Furthermore, it becomes the Rayleigh distribution function when $\beta = 2$ and reduces to the exponential distribution function when $\beta = 1$. Figure 1 shows the Weibull PDF for several values of the shape parameter when $\alpha = 1$ [8].

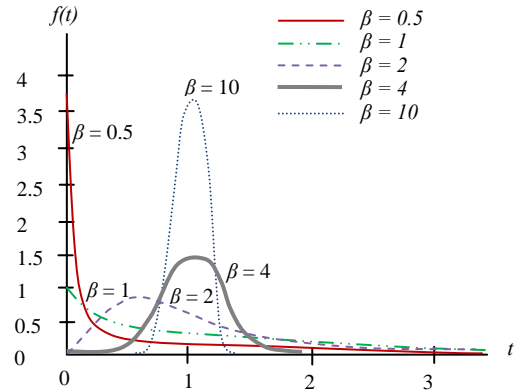


Figure 1. Weibull PDF for several shape values when $\alpha = 1$.

The reliability function of Weibull distribution is [15]:

$$R(t) = e^{-(t/\alpha)^\beta} \quad (2)$$

III. BUG REPORT ANALYSIS

The approach to the reliability estimates of the selected OSS projects consists of three steps: *bug-gathering*, *bug-filtering*, and *bug-analysis*. In the bug-gathering step, the online bug-repository systems are used to collect the failure data. Quality estimation of an OSS product depends on sufficient error reports and the accuracy of reports provided by the customers using the product. Although the collected reports may differ among bug-repository systems, the following fields are often common:

- Bug-ID - A unique identification for each bug reported.
- Bug-Time - The actual time of bug reported.
- But-Resolution – The state of a bug determined by the organization, such as whether the bug reported is valid, deleted, or fixed.
- Bug-Reporter – Information about the user submitting the bug report.
- Product – The specific product name or component in which a bug is detected.

Bugs reported may be duplicates, provide incomplete information, or may not represent real defects. Therefore, during the bug-filtering, such noises are removed from the bugs gathered in the first step. Finally, in the third step, the filtered data is organized into bug-frequencies for fixed time periods, which are set at two weeks. Observing the volume of bug-reports and the varying years of operations among the products, the choice of two-week periods provides sufficient number of failure frequencies that captures the pattern of bug reports. The bug-frequencies are then plotted against the time periods for further analysis in terms of graph fitting, comparison among the products, and reliability.

A. Bug Gathering and Filtering

The duration for which the failure data is collected for the five OSS projects are listed in Table II.

TABLE II. DURATIONS OF COLLECTED FAILURE DATA¹

Project name	Start date	End date
Firefox	03/1999 ²	10/2006
Eclipse	10/2001 ³	12/2007
Apache 2	03/2002	12/2008
ClamWin Free Antivirus	03/2004	08/2008
MPlayer	09/2002	06/2006

Table III illustrates a portion of a bug report stored at Sourceforge.net bug tracker. In the table, “Request ID” is Bug-Id; “Summary” is a short description of the problem reported; “Open Date” is the same as Bug-Time; “Priority” shows the level of bug severity, which is a value in the inclusive range of 1 – 9; “Status” is Bug-Resolution; and

¹ The start date of collected bug reports is the earliest date wherein a bug is reported.

² The failure data collected prior to the official release date of Firefox are obtained from Mozilla bug reports.

³ This date is prior to the official release date.

“Assigned To” is the person or the team that will be assigned to look into this specific bug. The “Status” field may carry different values such as “Open”, “Closed”, or “Deleted”. The “Open” value is an indication that the bug has not been fixed yet; “Closed” reflects that the bug has been fixed, and “Deleted” means that the bug reported is not a valid one.

TABLE III. A PORTION OF A BUG REPORT AT SOURCEFORGE.NET

Request ID	Summary	Open date	Priority	Status	Assigned to
1570891	can't set a playback position in FLV videos	2006-10-04 18:43	5	Open	Nobody
1565875	mencoder mp3lame option	2006-09-26 17:39	5	Open	Nobody
1565868	mencoder -oac copy	2006-09-26 17:28	5	Open	Nobody
1564167	Use second monitor bug	2006-09-23 18:16	5	Open	Nobody
1531952	screen resizing	2006-07-31 19:37	5	Open	Nobody

The bug reports extracted from Bugzilla are initially in XML format. A Java program is developed to gather the relevant data from the XML format for further data filtering and analysis. Figure 2 shows a portion of a bug report in XML stored at Bugzilla.

```

<bug>
  <bug_id>366101</bug_id>
  <creation_ts>2007-01-05 16:41 PST</creation_ts>
  <short_desc>nsIFile initWithPath should accept
  &quot;c:/mozilla&quot;; as native path (forward slashes should be treated
  as backslashes)</short_desc>
  <delta_ts>2007-01-05 16:57:22 PST</delta_ts>
  <reporter_accessible>1</reporter_accessible>
  <cclist_accessible>1</cclist_accessible>
  <classification_id>3</classification_id>
  <classification>Components</classification>
  <product>Core</product>
  <component>XPCOM</component>
  <version>Trunk</version>
  <rep_platform>PC</rep_platform>
  <op_sys>Windows XP</op_sys>
  <bug_status>NEW</bug_status>
  <priority>--</priority>
  <bug_severity>normal</bug_severity>
  ...
  <who name="David Hyatt">hyatt@mozilla.org</who>
  <bug_when>2000-04-13 16:16:07 PST</bug_when>
  ...
  <bug_status>VERIFIED</bug_status>
  <resolution>WORKSFORME</resolution>
</bug>

```

Figure 2. A portion of a bug report in XML at Bugzilla.

The fields in Figure 2 are self explanatory, like “Bug_id” is Bug-ID and “Who_name” is Bug-Reporter. The “Resolution” field is Bug-Resolution, which may hold different values:

- FIXED – This bug is a valid one and it is fixed.
- INVALID – This bug was not a valid one.
- WONTFIX – This bug will not be fixed.
- WORKSFORME – This bug cannot be reproduced.
- INCOMPLETE – Information provided is not sufficient for producing the bug.
- LATER – This bug will not be fixed in this version.
- REMIND – This bug will not probably be fixed in this version
- DUPLICATE – This is a duplicate bug reported earlier.

The bug reports collected are then filtered out. For the two projects from Sourcefoge.net, the bug reports with the status value of “Deleted” are discarded. For the failure data collected from Bugzilla, those bug reports with the following “Resolution” values are accepted and the rest are discarded: FIXED, WONTFIX, LATER, and REMIND.

B. Bug Analysis

In the bug-analysis step, the frequency of bugs in two-week periods is calculated. Therefore, the x-axis and y-axis represent the biweekly time and the corresponding bug frequency, respectively. For instance, x-axis in Figure 3 contains 116 points, which is equivalent to about 4.4 years of collected failure data for ClamWin operation. Figures 3-7⁴ show the bug frequency plots for the five OSS projects.

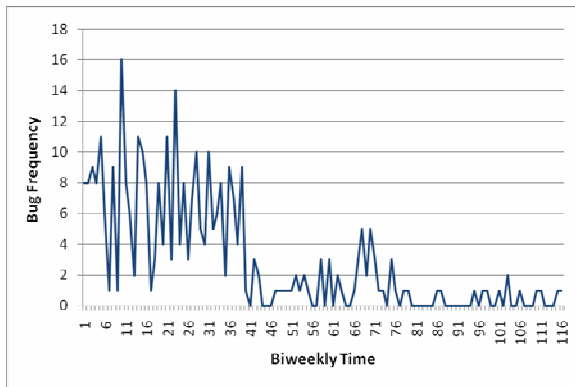


Figure 3. Filtered bug frequency for ClamWin Free Antivirus project.

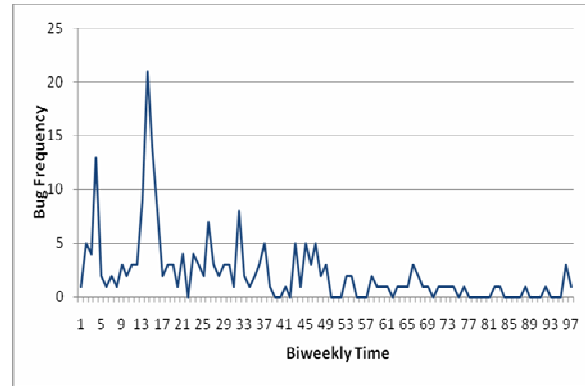


Figure 4. Filtered bug frequency for MPlayer OS X project.

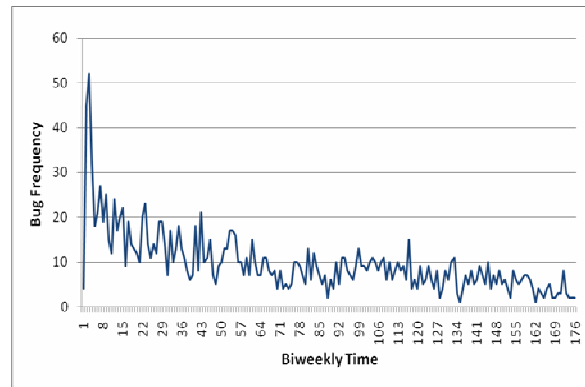


Figure 5. Filtered bug frequency for Apache 2 project.

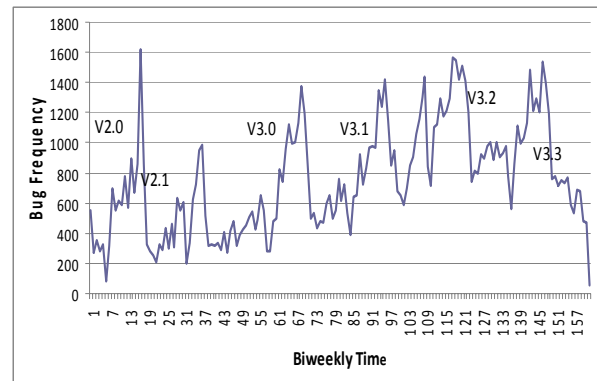


Figure 6. Filtered bug frequency for Eclipse project.

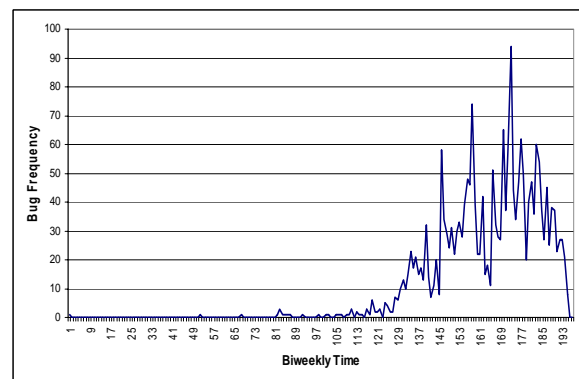


Figure 7. Filtered bug frequency for Firefox project.

⁴ In these figures and the follow up figures, the frequencies of bug reports are connected to form smoother plots. The purpose is to better visualize the trend of bug reports.

It is observed that the bug frequencies for three of these projects, i.e. Apache 2, MPlayer OS X, and ClamWin Free Antivirus, appear to follow a pattern that can be represented by the Weibull distribution function. As an example, Figure 8 shows this pattern that is visually superimposed on the bug frequencies for Apache 2.

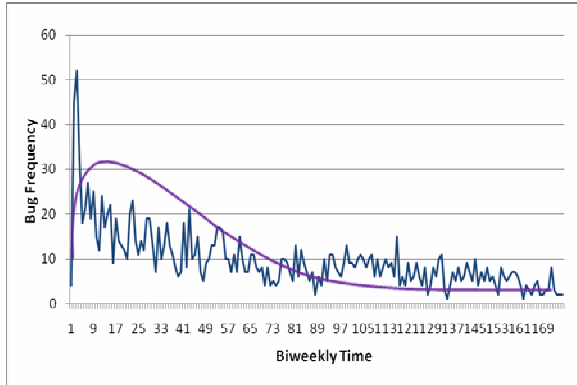


Figure 8. A curve fitted onto bug frequencies for Apache 2.

This pattern is supported by large body of empirical studies in that software projects follow a life cycle pattern described by Rayleigh distribution function, a special kind of Weibull distribution with shape parameter $\beta = 2$. This is considered 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 [8]. This pattern also is supported by Musa-Okumoto model in that the simple bugs are caught easily at the beginning of testing phase. The remaining bugs tend to be more difficult to detect because, for example, they are not exercised frequently. Therefore, the rate of undetected bugs drops exponentially as testing continues [9].

On a quick glance at Figure 6, Eclipse does not seem to follow this pattern. When bug reports are scrutinized in more detail, it is found that the versions separately follow a similar pattern. Figure 9 illustrates the bug frequencies for individual Eclipse releases superimposed in one diagram, rather than lumping the bug frequency for all versions as in Figure 6.

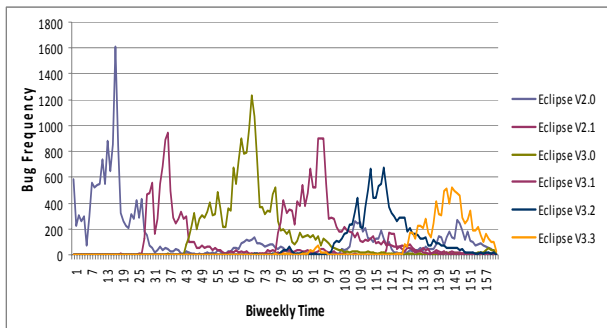


Figure 9. Filtered bug frequencies of Eclipse for different versions.

In this figure, the peak in each version shows the official release time for that version. In reality, the users started using the unofficial (beta) version and reporting the bugs before the official release of the version. For example, Eclipse V2.1 is officially released in March of 2003, but customers started downloading it in February 2003. In other words, the bug reports for version 2.1 are mixed with those of an earlier version. When the bug reports are correctly extracted based on the version numbers, each version seems to follow the same pattern as those of Apache 2, MPlayer OS X, and ClamWin Free Antivirus. A similar argument could be true for Firefox because of multiple peaks in Figure 7. However, the Firefox bug reports lack the version numbers. As seen in Figure 2, the bug reports contain the phrase “Trunk” for all versions. Therefore, the different versions of Firefox are treated as one unified version.

Figure 10 shows the bug frequencies for Eclipse V2.0 extracted from Figure 9. It is this version that will be used in reliability analysis of Eclipse because of its high bug reports in comparison to other versions. However, all versions of Eclipse will later be compared against each other.

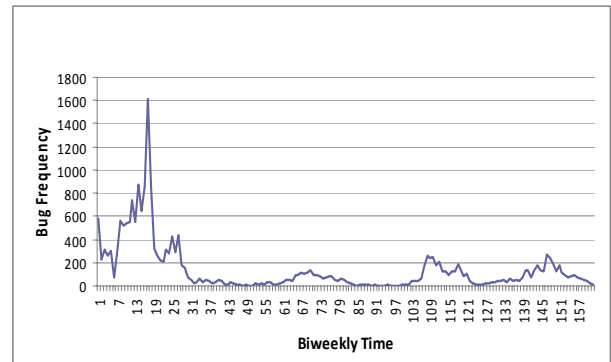


Figure 10. Filtered bug frequencies of Eclipse V2.0

The R Project is a freely available package that is used for a wide variety of statistical computing and graphics techniques. R is able to apply the Maximum Likelihood Estimation (MLE) technique [16] for estimating the parameters of Weibull distribution. Since R requires time-domain data, the relative frequency of bug reports needs to be converted to occurrence times of failure. Therefore, each bug report is mapped to its corresponding biweekly period. For example, 4 bugs reported in the 1st biweekly and 3 bugs reported in the 2nd biweekly periods are converted to: 1,1,1,1,2,2,2. This further illustrates that the total number of failures at the k^{th} position in the list is k , which implies that the input provided to R is cumulative.

The computed shape and scale for each OSS product is listed in Table IV. As indicated previously, the effect of the scale parameter is to squeeze or stretch the PDF graph. The greater the value, the greater the stretching. Therefore, the larger the value, the flatter the curve of PDF will be, which implies lower rate of failure. The correlation coefficient is a measure of strength of the match between the fitted curve and the bug frequencies pattern. The closer the coefficient value is to one, the stronger the match is.

TABLE IV. PARAMETER ESTIMATES FOR SELECTED PRODUCTS

Project name	Scale	Shape	Correlation Coefficient
Apache 2	67.93	1.17	0.95
Firefox	172.68	10.77	0.99
MPlayer	31.5	1.33	0.99
ClamWin Antivirus	31.36	1.23	0.99
Eclipse V2.0	49.88	0.94	0.90
Eclipse V2.1	61.06	1.72	0.91
Eclipse V3.0	76.82	3.64	0.91
Eclipse V3.1	91.14	6.14	0.93
Eclipse V3.2	120.19	8.99	0.96
Eclipse V3.3	144.82	15.74	0.94

Figure 11 shows the fitted graphs for the five projects. From the figure, the estimated PDFs are good representations of failure behavior. Among these, Eclipse versions have the lowest coefficient values. After some experimental analysis, the reason is due to a sharp increase of bug reports over a short period of time (about 10 biweeklies) in comparison to the measurement scale, which is about 160 biweekly periods. Since the increase and span of failures are correspondent to the shape and scale parameters, respectively, the estimates are favored toward the scale to cover a larger span of time. This implies the peak of the fitted graphs were pushed down, cutting off a portion of bug reports.

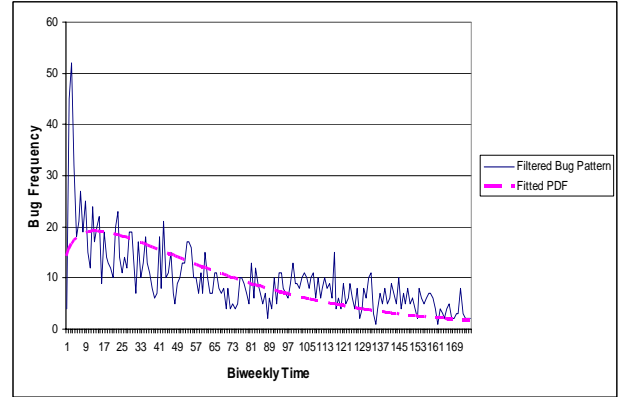


Figure 11c. Estimated PDF estimation for Apache project.

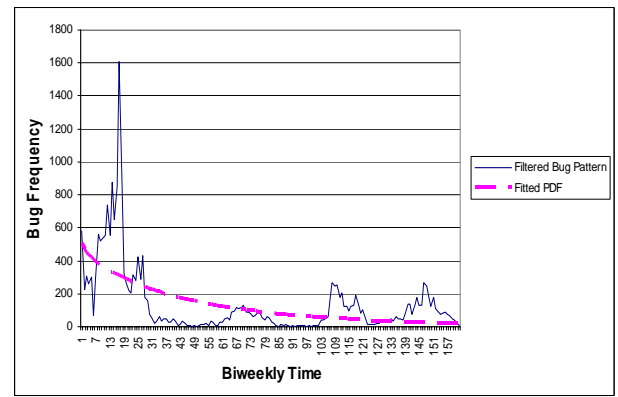


Figure 11d. Estimated PDF for Eclipse V2.0.

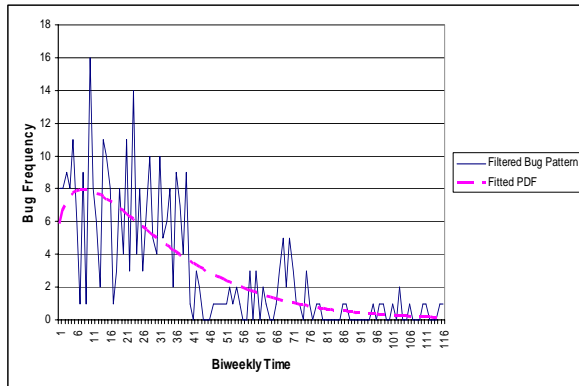


Figure 11a. Estimated PDF for ClamWin Free Antivirus project.

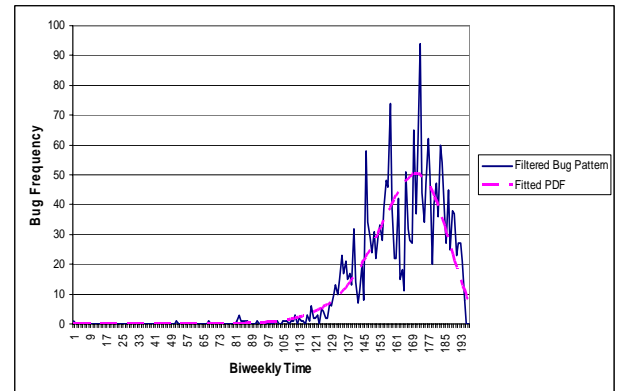


Figure 11e. Estimated PDF estimation for Firefox.

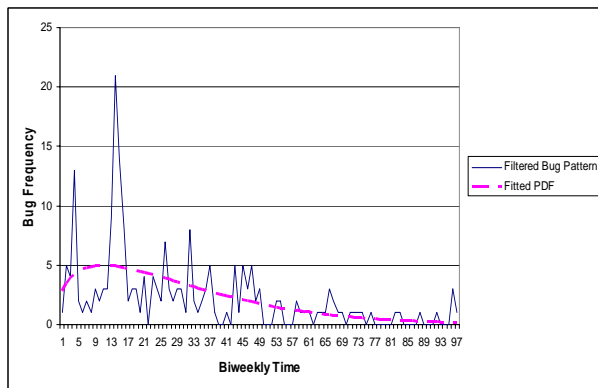


Figure 11b. Estimated PDF estimation for MPlayer project.

Reliability of the projects can be calculated by inserting the shape and scale parameters from Table IV into the Weibull reliability function in (2). Figure 12 exhibits the reliability graphs for the five OSS products. They all show a reliability of 0.90 and 0.80 over 4 and 8 biweekly periods, which are about 2 and 4 months of operation, respectively. Among these products, Firefox has the highest reliability. More accurate reliability estimate of Firefox would have been possible if version numbers were available. On the other hand, the figure shows that MPlayer, which almost completely overlaps ClamWin, has the least reliability. After

MPlayer and ClamWin, the graph of Eclipse V2.0 shows the next lowest reliability. Eclipse V2.0 could have performed worse if it were not for the future releases. More specifically, with a new release, the customers opt not to use the older versions. Hence, the number of downloads and bug reports would be reduced. Although this may not change the shape of the graph, it would likely reduce the steepness of the graph.

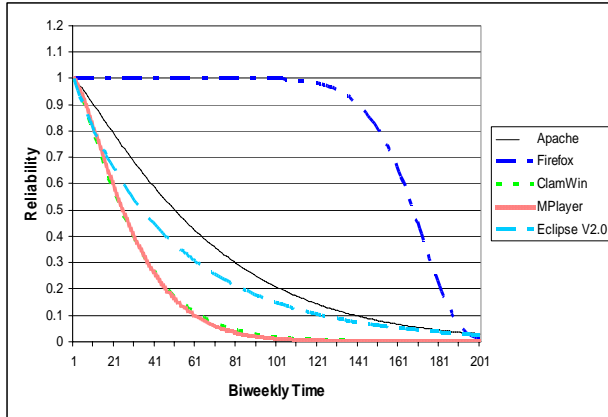


Figure 12. The reliability diagrams for the five OSS products

Table V shows the individual reliabilities for different time periods for each OSS project.

TABLE V. RELIABILITY ESTIMATES FOR VARIOUS BIWEEKLY PERIODS

Project / Biweekly periods	5	10	15	20	25
Apache 2	0.95	0.90	0.84	0.79	0.73
Firefox	1.00	1.00	1.00	1.00	0.99
MPlayer	0.92	0.80	0.69	0.58	0.48
ClamWin Antivirus	0.90	0.78	0.67	0.56	0.47
Eclipse V2.0	0.89	0.80	0.72	0.65	0.59

Figure 13 compares the reliabilities for different versions of Eclipse. As expected, as higher versions are released their reliabilities are improved. The latter three versions show a reliability of at least 0.90 for up to 60 biweekly periods, which is over two years of operation. The figure further indicates that the latest version, i.e. V3.3, has a reliability of 0.99 for 108 biweekly periods.

IV. CONCLUSION

Bug reports are the source for deeper understanding of failure distribution, classifying failures, and building accurate dependability models. The quality of bug analysis heavily depends on comprehensive and accurate recording of bug reports. The lack of a commonly accepted data format for archiving bug reports and efficient algorithms for data filtering adds to the complexity of failure data analysis.

In this experiment, two different trends of bug reports for the five OSS projects are visually discernible. The first group of products, i.e. Apache 2, MPlayer OS X, and ClamWin Free Antivirus, show the trend of a peak followed

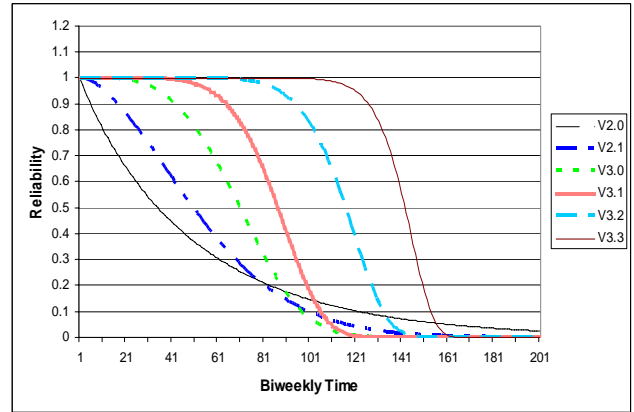


Figure 13. The reliability diagrams for different versions of Eclipse OSS

by a long decreasing tail, which is a good candidate to be fitted by the Weibull distribution function. The second group, which encompasses Eclipse and Firefox, shows multiple peaks instead of one particular peak. These products are more complex to analyze and to be modeled. The complexity can be explained by different factors such as coinciding of multiple versions and enclosure of different components or partial products in the main product. Although, it is likely that a PDF can be fitted, the chances for lower coefficient values for such products is higher, and the reliability growth estimates can be obscured. On the other hand, estimating reliability with least overlap among versions can provide a good sense of software quality over a period of time.

Although higher reliability is provided by later versions, care must be exercised when comparing them against products started from scratch. This is due to the fact that the reliability of later versions could be bloated as the result of borrowing many components from the earlier versions with detected and fixes of many prior defects. When analyzing Eclipse, the bug reports include the version numbers so that their associated bug reports can be extracted. But the same could not be said about Firefox. Hence, Firefox was treated as one composite version.

Because of time invested in collecting and mining the failure data, one avenue of future research is to investigate the reliability growth of these products with respect to some models of Non-Homogeneous Poisson Process (NHPP). The main issue in NHPP models is the determination of an expected value for the number of failures up to a point in time [6], [16].

REFERENCES

- [1] Bugzilla, <http://www.bugzilla.org>.
- [2] R.C. Cheung, "A user-oriented software reliability model", IEEE Transactions on Software Engineering, vol. 6, no. 2, March 1980, pp. 118-125.
- [3] J.De.S Coutinho, "Software reliability growth". IEEE Symposium on Computer Software Reliability, 1973, pp. 58-64.
- [4] A.L. Goel and K. Okumoto, "A time-dependent error-detection rate model for software reliability and other performance measure", IEEE Transactions on Reliability, vol. R-28, 1979, pp. 206-211.

- [5] S.S. Gokhale, M.R. Lyu, and K.S. Trivedi, "Reliability simulation of component-based software systems", Proceedings of 9th Int'l Symposium on Software Reliability Engineering, 1998.
- [6] IEEE Reliability Society, "IEEE recommended practice on software reliability", IEEE Std 1633-2008, June 2008.
- [7] Z. Jelinski and P.B. Moranda, "Software reliability research", in Statistical Computer Performance Evaluation, W. Freiberger, Ed., New York: Academic Press, 1972, pp. 465-484.
- [8] H.S. Kan, Metrics and Models in Software Quality Engineering, 2nd Ed., Addison-Wesley, 2003.
- [9] I. Koren and C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann, 2007.
- [10] B. Littlewood and J.L. Verrall, "A bayesian reliability model with a stochastically monotone failure rate", IEEE Transactions on Reliability, vol. R-23, June 1974, pp. 108-114.
- [11] B. Littlewood and J.L. Verrall, "A bayesian reliability growth model for computer software", Applied Statistics, vol. 22, 1973, pp. 332-346.
- [12] A. Mockus, T.R. Fielding, and J.D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology, vol. 11, no. 3, July 2002, pp. 309-346.
- [13] T.R. Moss, The Reliability Data Handbook, ASME Press, 2005.
- [14] 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.
- [15] K. Neubeck, Practical Reliability Analysis, Prentice Hall, 2004.
- [16] H. Pham, Software Reliability. Springer-Verlag, 2000.
- [17] E.S. Raymond, "The cathedral and the bazaar: musings on linux and open source by an accidental revolutionary, 2nd Ed., O'Reilly, 2001.
- [18] R Project, <http://www.r-project.org/>.
- [19] M.L. Shooman, "Probabilistic models for software reliability prediction", in Statistical Computer Performance Evaluation, W. Freidberger, Ed., New York: Academic Press, 1972, pp. 485-502.
- [20] SourceForge, <http://sourceforge.net>.
- [21] Y. Zhou and J. Davis, "Open source software reliability model: an empirical approach", Proceedings of the 5th Workshop on Open Source Software Engineering, May 2005, pp. 1-6.
- [22] W.L. Wang, Y. Wu and M.H. Chen, "An architecture-based software reliability model", Proceedings of Pacific Rim Int'l Symposium on Dependable Computing, 1999.
- [23] S. Yacoub, B. Cukic and H.H. Ammar, "A software-based reliability analysis approach for component-based software", IEEE Transactions on Reliability, vol. 53, no. 4, Dec 2004.
- [24] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection", IEEE Transactions on Reliability, Vol. R-32, 1083, pp. 475-478.