Modelling Failures Occurrences of Open Source Software with Reliability Growth

Bruno Rossi, Barbara Russo, and Giancarlo Succi

CASE – Center for Applied Software Engineering
Free University of Bolzano-Bozen
Piazza Domenicani 3, 39100 Bolzano, Italy
{brrossi, brusso, gsucci}@unibz.it
http://www.case.unibz.it

Abstract. Open Source Software (OSS) products are widely used although a general consensus on their quality is far to be reached. Providing results on OSS reliability - as quality indicator – contributes to shed some light on this issue and allows organizations to make informed decisions in adopting OSS products or in releasing their own OSS. In this paper, we use a classical technique of Software Reliability Growth to model failures occurrences across versions. We have collected data from the bug tracking systems of three OSS products, Mozilla Firefox, OpenSuse and OpenOffice.org. Our analysis aims at determining and discussing patterns of failure occurrences in the three OSS products to be used to predict reliability behaviour of future releases. Our findings indicate that in the three cases, failures occurrences follow a predetermined pattern, which shows: a) an initial stage in which the community learns the new version b) after this first period a rapid increase of the failure detection rate until c) very few failures are left and the discovery of a new failure discovery is rare. This is the stage in which the version can be considered reliable.

Keywords: Software failures, software reliability growth, open source software.

1 Introduction

Many of the open source projects do not have resources to dedicate to accurate testing or inspection so that the reliability of their products must rely on community's reports of failures. The reports are stored in the so-called bug tracking systems, are uploaded by the community, and moderated by internal members of the open source project. Reports are archived with various pieces of information including the date of upload and the description regarding the failure. What information can be collected from these repositories and how to mine them for reliability analysis is still an open issue ([6], [5]).

This paper proposes a method to mine bug repositories in order to determine patterns of failure occurrences that can be used to model reliability of past, current, and future versions of an open source product. In particular, this work discusses

whether the traditional theory of software reliability growth can be readily applied to data coming from open source products. Our approach relies on an deep understanding of the bug tracking system used by each open source project and an accurate cleaning of the data. Specifically, we have examined the user reports across versions of three open source projects, Mozilla Firefox[1], OpenOffice.org[2], and OpenSuse[3]. For each project and version, we have extracted information on the dates of issue opening and the number of open reports per day. We have fitted the traditional software reliability growth models with the time series of failures occurred per day for each version. We have repeated this procedure for major versions of the product and we have ranked the resulting best fit models for each version by a set of measures of model accuracy and reliability prediction. To understand whether there is a pattern in the failure occurrences of a product, we simply counted the number of times a model type is ranked at the top among all the versions and per measure of accuracy or prediction.

The paper is organized as follows. Section 2 consists of the background and some literature related to the work. This section briefly introduces to Software Reliability Growth Models and measures of accuracy and prediction. Section 3 illustrates the data sample and reviews the assumption we made in data collection and analysis. Section 4 introduces the method of regression across versions and pattern definition. Section 5, reports of the findings. Section 6 illustrates the limitation of the work and the future work. With section 7 we conclude.

2 Background

Modelling failure occurrences with Software Reliability Growth (SRG) is a classical approach in static analysis for software reliability ([2], [10], [11], [14]). In reliability growth, failure occurrences are assumed to grow with time such that the time to the next failure increases with time, too. This behavior is modeled with stochastic processes that describe the cumulative number of failures over time ([13]). The expected mean of the stochastic process defines a parametric function of time and represents the expected total number of failures at any instant of time t. Then the parameters of the expected mean are determined by non-linear regression on the actual dataset of cumulative number of failures over time.

The major challenges in SRG is to determine the mathematical expression of the expected mean either with a constructive procedure or with regression on predetermined families of time curves. While the former requires a deep understanding of the failures occurrence process in the operational environment of the software product ([10]), the latter focuses on an ex-post study shifting the complexity of the analysis to the data cleaning, model fitting, and interpretation, ([8], [9]). In this paper, we adopt the latter approach for the type of information we were able to gather with remote queries to data of the open source repositories.

[1] http://www.mozilla-europe.org/en/firefox
[2] http://www.openoffice.org
[3] http://www.opensuse.org/en

Although this approach seems more feasible in our case, mining repositories for software reliability is not straightforward. To understand this we need to recall the meaning of software reliability. Software reliability refers to the capability of a software system to follow given specifications in a given interval of time and in given *operational* settings. A system experiences a failure when it deviates from this behaviour during its usage. As such, the best indicators for failures are users' reports on misbehaviour of the system during its usage. Gathering data on usage misbehaviour is hard as it depends on users' feedback, which is difficult to trace down. As a consequence, the majority of the classical works in SRG has used the same failures databases[4], not publicly accessible ones, or databases of defects discovered during internal testing ([10]) preventing proper conclusions on real system misbehaviour and replications in different applicative domains ([4]).

Nowadays, open source projects represents a new source of data for software reliability. They are as open to the everyone and provide enough information to perform analysis replications, but they may contain duplicate information, lack of complete information, and use localizations of standards and terminology. For example, in our case we were not able to find in the repositories information on deployment, customer usage profiles, and testing data that have been proved to be good predictor of failures occurrence rates and failures occurrence patterns for software systems prediction, ([2]). We also had to discard some open source projects that apparently had a large amount of users' report, but that at the end - after cleaning – data was too scarce to perform a rigorous analysis. Even with such drawbacks, we still believe that reliability analysis on open repositories is of great value as it is based on the real user's reports the effort is to understand whether the specific source provides suitable information.

Among several studies that analyzed open source software from the point of view of software reliability, we can cite ([12], [15], [18]). For example, Eclipse, Apache HTTP Server 2, Firefox, MPlayer OS X, and ClamWin Free Antivirus applications have been evaluated by means of several models, and the Weibull distribution has been found to adapt well in modelling simpler projects, although more complex models are claimed to be needed for Firefox and Eclipse [12]. In [15], authors are interested in evaluating the reliability of a complex system developed with a distributed approach, the Xfce desktop. Using decision support methods and non-homogeneous Poisson processes (NHPP), authors show how to model the complex interactions among components for failure reliability prediction. In [18], several open source software projects have been analyzed and found to behave similarly to equivalent closed source applications. Also in this case, the Weibull distribution has been found to be a simple and effective way to represent software reliability growth.

In our research, after selecting the appropriate open source projects and cleaning the data our work provides an extension of the work of Li et al ([7]) and Succi et al. ([14]) combining the two approaches - software reliability growth across versions ([7]) and reliability growth and measures of accuracy and prediction ([14]) - and replicating the study on three different OSS. Mining repositories for software failures

[4] Like the PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada http://promise.site.uottawa.ca/SERepository/ or the new one http://promisedata.org/

is a hot topic ([1], [2], [5], [6], [7], [8]), but at our knowledge it does not seem to be any research that has compared results across multiple releases and multiple OSS through multiple measures of accuracy and prediction.

2.1 Candidate Software Reliability Growth Models

We use the Software Reliability Growth Models (SRGMs) adopted in ([14]). SRGMs are stochastic processes described by a counting random variable determined by the cumulative number of failures over time ([13]). The expected mean at a given time t of these models is defined by a parametric curve in t, $\mu(t)$. The goal of software reliability growth is to determine the parameters of the expected mean. Many of the SRGMs we used are Poisson Processes whose mass probability is defined by a Poisson distribution. The basic model of them is the Musa exponential model ([11]) whose expected number of failures is defined by the parametric expression:

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

SRGMs can be S-shaped or concave depending whether they change concavity at least once. S-shaped models have an initial learning stage in which the failures detection rate, starting from very low values, shows a first increase and then a decrease that finally approaches to zero. Interpreting the model with time between failure occurrences, S-shaped models show an initial slow pace of failures occurrences, then a period in which failures are reported frequently until when relatively few failures remains in the code and failure discovery becomes hard. Example of S-shaped models are Weibull and Hossain Dahiya models. Concave models do not foresee such curve and reveal a good knowledge of the new released version. Failures are discovered soon after the release with a fast pace. Example of concave models are the Goel- Okumoto and Gompertz models.

In addition, models can be finite or infinite. Finite models have an horizontal asymptote and therefore a finite total expected number of failures are assumed in the product. The interested reader can find further details in the papers of Succi *et al.*, ([14]), and in the book of Lyu, ([10]).

2.2 The Measures of Accuracy and Prediction

The measures of accuracy and prediction capture the properties of a best-fit SRGM. We group the measures two by two by the attribute of the model we want to investigate. Table 1 introduces them indicating the references for further readings.

Table 1. Measures of accuracy and prediction used to rank the best fit models.

Model Attribute	Measure	Reference
Goodness of fit	Coefficient of Determination (R2)	[3]
	Akaike Information Criterion (AIC)	[7]
Precision of fit	Relative precision to fit (RPF)	[14]

	Coverage of fit (COF)	[16]
Forecasting ability	Predictive ability (PA)	[14]
	Accuracy of the final point (AFP)	[17]

Goodness of fit and precision of fit refers to modelling the dataset, whereas forecasting ability defines the prediction nature of the model. Goodness of Fit expresses the ability of a mathematical model to fit a given set of data (R^2 and AIC). Precision of fit provides the extent (RPF) and the ability of the model to capture the data (COF) in the 95% confidence interval of the model. In general, these two measures are used in combination as they give complementary information on model precision. Measures of forecasting ability define the capability of a model to predict early in time (PA) or accurately (AFP) the final total number of failures.

3 The Dataset

We have selected three well-known OSS products: a web browser, Mozilla Firefox, an office suite, OpenOffice.org, and an operating system, OpenSuse. Each of these products maintains a bug tracking system open to the community based on Bugzilla[5]. We have chosen these three software projects also because their project's strategy for failures storage and the terminology used in Bugzilla is enough similar.

For each version found in the bug tracking system we have collected all the issues reported at our date of observation together with the date at which they were reported (date of opening). For each open source project, we have considered all the major versions until mid 2008 with more than 40 failures. For OpenOffice.org we were able to get 13 versions until 2006 (that we decided they were enough for the purpose of this analysis). Unfortunately, OpenSuse and Mozilla Firefox had not so many reports and versions at the time of our data collection and we had to limit the versions to five for OpenSuse and three for Mozilla Firefox.

Table 2 illustrates the dataset collected and the models used on the datasets.

Table 2. The Datasets.

Application	SRGMs type[6]	Versions	Time Window
Mozilla Firefox	Weibull, Goel Okumoto, Gompertz, Logistic, Goel Okumoto Sshaped, Hossain Dahiya	1.5, 2.0.0.0 and 3.0.0.0	23.10.2006-8.6.2008
OpenOffice.org	Goel-Okumoto, Goel-Okumoto S-shaped, Weibull, Hossain-Dahiya, Yamada	1.0.0, 1.0.1, 1.0.2, 1.0.3, 1.1.0, 1.1.1, 1.1.2, 1.1.3, 1.1.4, 2.0.0, 2.0.1, 2.0.2, 2.0.3	16.9.2001-21.08.2006

[5] http://www.bugzilla.org

[6] The mathematical expression of the SRGMs model can be found in ([10]) or in ([13])

OpenSuse	Goel-Okumoto, Goel-Okumoto S-shaped, Weibull, Hossain-Dahiya, Gompertz, Logistic	10.0,10.1,10.2,10.3, 11.0	5.9.2005-1.6.2008

The Bugzilla repository allows to mine failures with sophisticated queries. A report in Bugzilla is called issue or bug and it may carry a large amount of information that needs to be pruned according to the research that is performed. To choose the view more appropriate for the research objective, one needs to understand in details the life cycle of an issue. In Fig. 1, we report the standard life cycle of a report submitted to a Bugzilla repository as described in the documentation[7].

Although the three projects we have chosen are all supported by a Bugzilla repository, we found out that their customization significantly varies. As such, starting from the description in Fig. 1., we had to put some further effort in the homogenization of the terminology and the procedures used by the open source projects in moderating the issues.

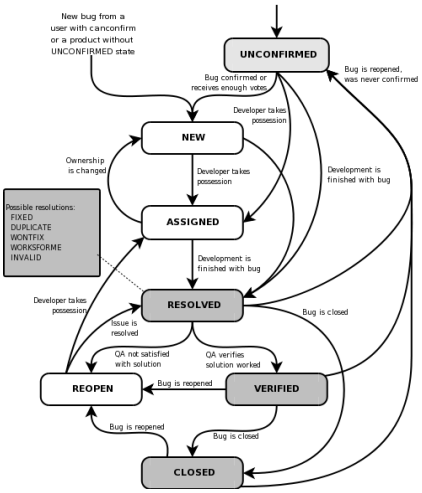

Fig. 1. Bugzilla life cycle of an Issue[7].

3.1 Empirical Assumptions

In this section we discuss all the assumptions we have made on the data before starting our analysis.

Looking at the cumulative number of failures, we have noticed that in each version the rate of growth tends to zero for large values of time (Fig. 2, 3, and 4) indicating a

[7] http://www.bugzilla.org/docs/tip/en/html/lifecycle.html

bound for the total number of failures. For this reason, we have decided to include only finite models in our analysis, ([10]).

After a deep inspection of the repositories and of their documentation, we have decided to focus on those issues that were declared "bug" or "defect" excluding any issue that was called something like "enhancement," "feature-request," "task" or "patch". This would have guarantee that our analysis dealt with proper failures. For the same reason, we have considered only those issues that were reported as closed or fixed (according to the terminology of the single repository) after the release date of each version. Namely, reports before the release date were in general related testing release candidates and they were not expressing the reliability of the version.
For the same reason, we have cleaned the dataset from issues that were declared something like "duplicate," "won't fix," or "it works for me." Table 3 illustrates our choice.

Table 3. Chosen view of the Bugzilla repositories.

Issue Type	Status	Resolution	Platforms and operating systems
Defects/Bugs	Resolved, Verified, Closed, New, Started, Reopened	Fixed/Closed	All

4 The Method

For each version of Table 2, we have collected failures according to their date of opening. In this way, we had defined a time series of cumulative number of failures per version. The following three pictures illustrate a sample of the time series and the plots of their SRGMs. In Fig. 4 we also report the 95% confidence interval of the Hossain Dahiya model.

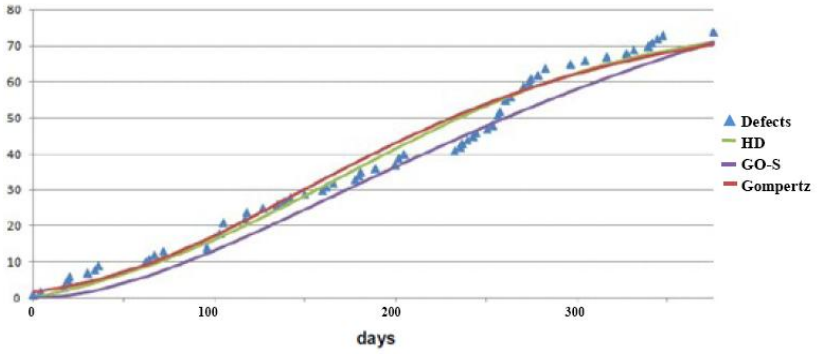

Fig. 2. Cumulative number of failures of Firefox version 1.0, the best-fit model (HD) and its 95% confidence interval. HD: Hossain Dahiya, GO-S: Goel Okumoto S-shaped, Gompertz.

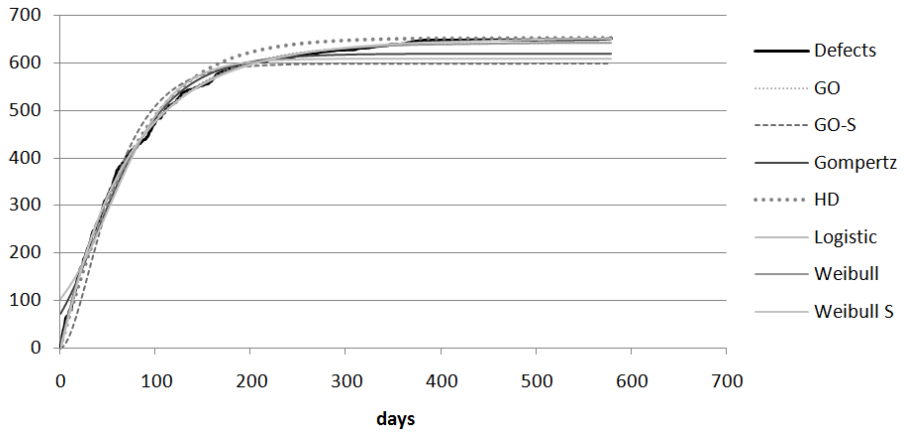

Fig. 3. Cumulative number of failures of OpenSuse version 10.0 and its best-fit models. GO: Goel Okumoto, GO-S: Goel Okumoto S-shaped, Gompertz, HD: Hossain Dahiya, Logistic, Weibull S: Weibull S-shaped, Weibull more S: Weibull S-shaped, Yamada.

For every OSS chosen, we have fitted the parametric expected mean of the SRGMs with the actual cumulative series of failures per each version of Table 2. As a result we have obtained a set of best fit models each corresponding to one type of SRGM in Table 2. At the end, we have obtained 18 SRGMs (6 each version) for Mozilla Firefox, 52 SRGMs (4 each version) for OpenOffice.org, and 25 SRGMs (5 each version) for OpenSuse. Fig. 2, 3, and 4 illustrate the time series and the curves corresponding to the best-fit models for one version of the three OSS. We have ranked the resulting best-fit models of every version by the measures of model accuracy and reliability prediction of Table 1.

A SRGM that outperforms for a given measure across versions of an OSS represents a pattern of reliability for that OSS. To determine the pattern we have simply counted for each measure of accuracy or prediction of Table 1, the number of time a best fit SRGM of a given type outperforms across the versions. In the following section, we discuss the existence and the type of pattern we have found.

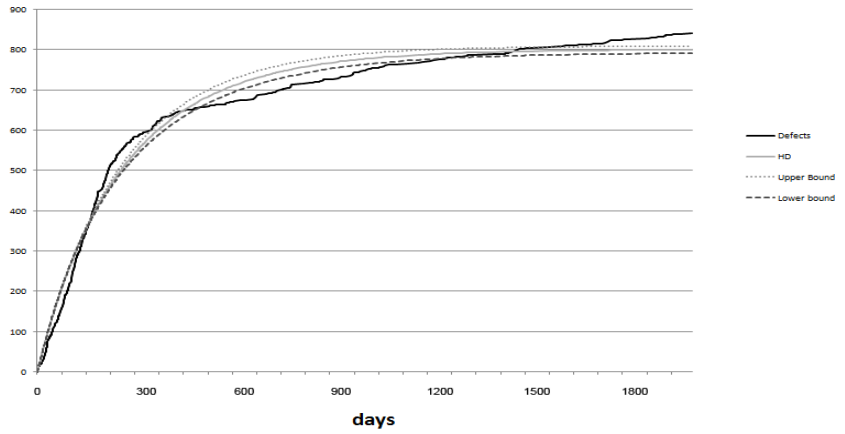

Fig. 4. Cumulative number of failures of OpenOffice.org version 1.0.1 and its best-fit models. HD: Hossain Dahiya.

5 Findings

In the following we report of the findings for the three OSS. The Yamada model is not included as its analysis does not report of significant results.

OpenSuse. For all the versions of OpenSuse, the Weibull model is the best one in all the measures but prediction ability. Thus, we can say that across versions it represents the data (Goodness of fit), it captures the majority of the data in small area of confidence (Precision of fit), and it is accurate in determine the final number of defects that partially determines forecasting ability. Namely it is not the best for prediction ability, thus we cannot use it to predict the total number of defects early in time. In fact, there are no SGRMs that can do it as no model outperforms across versions for predictive ability.

The predominance of the Weibull model confirms the findings in ([7]) extending the result to other measures of accuracy than the Akaike Information Criterion.

In Table 4 we report the rankings for version 11.0.

Table 4. Ranking of the best fit models for OpenSuse version 11.0.

Model Type	R Squared	AIC	CoF	RPF	AFP	PA
Weibull	**0.998**	**1.38**	**96%**	3075.47	**0,05**	0.77
Weibull S-shaped	0.997	1.4	**96%**	4057.85	0.30	0.77
Goel Okumoto	0.995	1.49	82%	792.31	1.91	0.77

Gompertz	0.993	1.6	18%	**144.02**	3.15	0.75
Logistic	0.985	1.75	62%	642.75	4.79	0.75
Goel Okumoto S-shaped	0.956	1.95	69%	10838.81	7.51	0.79
Hossain Dyohain	0.935	2.05	2%	N/A	9.64	**0.71**

Mozilla Firefox. The Weibull model dominates in all the versions for all the measures but accuracy of the final point and relative precision of fit. Again this confirms and extends the results in ([7]). In Table 5, we illustrate the values of the measures of version 1.5 of Mozilla Firefox. The values in boldface are the best ones. In this version, the Weibull model has the worst overall behaviour compared with the other versions. Nonetheless, it is in the top most ranking in three out of six measures and well performing in the rest. The exceptional case is RPF that measures the area of the 95% confidence interval over the time span of the series. As measure of accuracy RPF is subsidiary to CoF and it is used in combination with it. Thus, as models with low CoF are less relevant per se, the Weibull model is the most interesting, although the ratio CoF / RPF is not the highest. The Weibull model is not the best for prediction though. This is true for all the versions and in particular in the case of AFP for version 1.5 in Table 5.

Table 5. Ranking of the best fit models for Mozilla Firefox version 1.5.

Model Type	R Squared	AIC	CoF	RPF	AFP	PA
Weibull	**0.973**	2.55	**94%**	35.52	0.04	**0.62**
Goel Okumoto	**0.973**	**2.51**	56%	7	**0.022**	**0.62**
Gompertz	0.954	3.08	18%	**2.68**	0.13	N/A
Logistic	0.934	3.45	25%	6.43	0.15	N/A
Goel Okumoto S-shaped	0.956	3.02	37%	4.52	0.17	N/A
Hossain Dyohain	**0.973**	2.54	80%	13.02	0.024	**0.62**

OpenOffice.org. The Weibull model is again the best model across the measures of accuracy and prediction. In particular, for CoF the model outperforms 70% of the times across the versions.

The predominance of the Weibull model across so many versions as in OpenOffice.org is definitely significant.

Table 6. Ranking of the best fit models for OpenOffice.org version 2.0.

Model Type	R Squared	AIC	CoF	RPF	AFP	PA
Weibull	**0.998**	**3.08**	**0.95**	27.09	**0.001**	47%
Goel Okumoto	0.994	4.05	0.32	7.59	0.04	47%
Gompertz	0.98	5.24	0.17	**5.89**	0.07	**45%**
Logistic	0.965	8.85	0.24	23.86	0.09	**45%**
Goel Okumoto S-shaped	0.946	6.21	0.23	14.35	0.12	N/A
Hossain Dyohain	0.991	4.42	0.34	11.44	0.054	46%

Comparing the results for the three applications we can say that:

- For Goodness of fit (R square and AIC) the Weibull model confirms its superiority according to the work of Li et al. ([7]). The model well represents the data and the Weibull pattern can be used to represent future versions.
- For Precision of fit the 95% confidence interval of the Weibull model is the best in capturing data within its 95% confidence interval (CoF) although sometimes with a poor density (or equivalently in a large confidence interval, RPF). This is a measure of spread of data around the model also accounting for its variation in its confidence interval. Any significant variation of the model still represent the data with enough precision. This might turn to be useful when we discuss the time of occurrences in the open source repositories. The time reported in the repositories (calendar time) might not refer to the real time in which the user has been experiencing a failure. Some delay might have been occurred. As such, the result for Precision of fit even if it gives a positive answer for the Weibull model is not completely satisfactory and a Monte Carlo sensitivity analysis on the timing for each version will be matter of future work.
- For Predictive ability the Weibull model is definitely good to estimate the final total number of failures (AFP) but it cannot be used – as any other SRGM – for early prediction (PA). For this purpose, other approaches might be considered in combination ([1], [8], [9]). The lack of a pattern for PA further means that in the majority of the versions of the three OSS there is a low increase of the failure detection rate as in Fig. 2 (with or without a learning effect) so that it is impossible to predict the total number of failures of a version at the early stage of the failure reporting process. Thus, the case of Fig. 3 where a sudden and early increase of the detection rate appears, is less frequent across the versions of the three products Fig. 2 better represents the data as after 300 days the number of failures is still far to be near to the total final number of the version.

6 Limitations and Future Work

During our inspection, we have understood that the bug tracking system is regularly used by the internal team of the project. Internal team members know better the application so their reports might not represent typical end-user's reports of a failure. Although we have used some measure to limit this bias (section 4.1), as we could not differentiate between issues reported by the internal team and the rest of the world, we could not guarantee that the dataset is a dataset of failures reported only by end-users. In any case, as we have considered reports issued only after the release date, the reports of the internal team members refer to an operative period of the OSS and as such they contribute to some extant to the overall reliability of the OSS.

The date of report might be not exactly the date of failure discovery. There might have been some delay in reporting the issues and - depending on the repository – in the assignment of the date of opening during the moderation of the issue. This might create a noise in the timing and it will be matter of future research.

The number of versions, the number and types of SRGMs, and the time windows of the observations are different in the three OSS. This was due to some time constraints, the availability of the data in the repositories, and the missing values for the measures in Table 1. As we do not pretend to compare the three OSS, but we rather want to understand whether there is a pattern of reliability in each OSS, this difference is not crucial.

7 Conclusions

The goal of this paper was to present an approach to investigate reliability of OSS with software reliability growth. We used open on-line repositories to collect data of three different projects. We intensively cleaned the data we collected to limit the bias associated with the open nature of these repositories.

We found that the classical theory of software reliability growth is appropriate for such data and it is a good instrument to model failure occurrences across software versions.

We found that the Weibull model is the best model that fits the data across all the versions for each OSS (Goodness of Fit) with a low percentage of outliers (Precision of Fit). This confirms the results obtained in ([7]) for the Akaike Information Criterion and reveals a common pattern of software reliability for the three OSS. Namely, the Weibull model is the best SRGM that represents the failure occurrences in the three open source products. It is an example of S-shaped curve and as such it indicates an initial learning phase in which the community of end-users and reviewers of the open source project does not react promptly to new release. This slow pace at which failures are reported might originate from various causes like, for example, the unfamiliarity with the project or its complexity. Given the existence of candidate releases and intermediate versions one could expect that the community were ready to report of failures soon after the public release date. But the learning curve proves differently in three OSS.

Given the dominance of the Weibull model across the versions of an OSS we can assume that this type of model can be used for the future versions of the OSS. The open question is how to predict the parameters of this model without fitting the model on future data. In [8] the authors propose to use a combination of code and time measures. This approach will be matter of future research.

For OpenOffice.org and OpenSuse, the Weibull models can also be used for accuracy of the final point telling mangers when the version can be considered reliable as few failures remain to be discovered. As Fig. 2 shows, this does not hold for Mozilla Firefox. This may suggest how the community reacts differently to new releases of Firefox showing a slow pace to report failures in the early days after the release.

Yet, SRGMs are not a good instrument to early prediction of the total number of failures within the operational life of an OSS. Other instruments like Bayesian models, Product/Process models or genetic algorithms might be explored in combination with predictive ability ([1], [8], [9]).

Acknowledgments. We thank Sufian Md. Abu, Stefan Mairhofer, Gvidas Dominiskaus for their help in data collection and cleaning. We also thank the projects Mozilla Firefox, OpenSuse, and OpenOffice.org for the data supplied for this study.

References

1. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.: Is it a bug or an enhancement?: a text-based approach to classify change requests. CASCON 2008, Vol. 23 (2008)
2. Bassin, K., Santhanam, P.: Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. Eighth International Symposium on Software Reliability Engineering, Case Studies, IEEE Computer Society, pp. 103 – 114 (1997)
3. Draper, N.R., Smith, H.: Applied Regression Analysis. Wiley- Interscience (1998)
4. Fenton, N., Neil, M.: A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering, 25(5) pp. 675 – 689 (1999)
5. Godfrey, M. W., Whitehead, J.: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver Canada, May 16-17 (2009)
6. Li, P.L., Herbsleb, J., Shaw, M.: Forecasting field defect rates using a combined timebased and metrics-based approach: a case study of OpenBSD. 16th IEEE International Symposium on Software Reliability Engineering (ISSRE), pp. 10-19 (2005)
7. Li, P.L., Herbsleb, J., Shaw, M.: Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: a Case Study of OpenBSD. 11th IEEE International Symposium on Software Metrics (2005)
8. Li, P.L., Shaw, M., Herbsleb, J., Ray, B, Santhanam, P.: Empirical evaluation of defect projection models for widely-deployed production software systems. Proceedings of the twelfth international symposium on Foundations of software engineering, pp. 263 – 272 (2004)
9. Li, P.L., Shaw, M., Herbsleb, J.: Selecting a defect prediction model for maintenance resource planning and software insurance. Proceedings of the Fifth International Workshop on Economics-driven Software Engineering Research, Oregon, USA (2003)
10. Lyu, M.R.: Handbook of Software Reliability Engineering, McGraw-Hill (1996)

11. Musa, J.D., Iannino, A., Okumoto, K.: Software Reliability: Measurement, Prediction, Application, McGraw-Hill (1989)
12. Rahmani, C. Siy, H., Azadmanesh, A.: An Experimental Analysis of Open Source Software Reliability, F2DA 2009 Workshop on 28th IEEE Symposium on Reliable Distributed Systems, Niagara Falls (2009)
13. Rigdon, S. E., Basu, A. P.: Statistical Methods for the Reliability of Repairable systems, Wiley and Sons, pp. 281 (2000)
14. Succi, G., Pedrycz, W., Stefanovic, M., Russo, B.: An Investigation on the Occurrence of Service Requests in Commercial Software Applications, Empirical Software Engineering Journal, 8(2), pp. 197-215 (2003)
15. Tamura, Y., Yamada, S.: Comparison of Software Reliability Assessment Methods for Open Source Software, 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS'05), vol. 2, 488-492 (2005)
16. Wood, A.: Predicting software reliability. Computer, 29(11), pp. 69-77 (1996)
17. Yamada S., Ohba M., Osaki S.: S-Shaped Reliability Growth Modeling for Software Error Detection. IEEE Transactions on Reliability, December, pp. 475-484 (1983)
18. Zhou, Y., Davis, J.: Open Source Software Reliability Model: an empirical approach International Conference on Software Engineering: Proceedings of the fifth workshop on Open Source Software Engineering, St. Louis, MO (2005)