Reliability Growth in Software Products

Pankaj Jalote
Visiting Researcher
Microsoft Corporation, Redmond, USA
(pjalote@microsoft.com)

Brendan Murphy
Microsoft Research
Cambridge, UK
(bmurphy@microsoft.com)

Abstract

Most of the software reliability growth models work under the assumption that reliability of software grows due to the bugs that cause failures being removed from the software. While correcting bugs will improve reliability, another phenomenon has often been observed – the failure rate of a software product, as observed by the user, improves with time irrespective of whether bugs are corrected or not. Consequently, the reliability of a product, as observed by users, varies, depending on the length of time they have been using the product. One reason for this reliability growth is that as the users gain experience with the product, they learn to use the product correctly and find work-around for failure-causing situations. Another factor that affects this growth is that following the product installation, the user discovers that other actions may be required, like installing new drivers, upgrading other software to a compatible version, etc. to properly configure the new product. In this paper we present a simple model to represent this phenomenon – we assume that the failure rate for a product decays with a factor α per unit time. Applying this failure rate decay model to the data collected on reported failures and number of units of the product sold, it is possible to determine the initial failure rate, the decay factor, and the steady state failure rate of a product. The paper provides a number of examples where this model has been applied to data captured from released products.

P. Jalote's regular address is: Department of Computer Science and Engineering; I. I. T.; Kanpur – 208016; India; email: jalote@cse.iitk.ac.in

1. Introduction

Many software reliability growth models have been proposed, which estimate the reliability of a software system as it undergoes changes through the removal of failure causing faults. The focus of these models is often on the product behavior during system test, with the hope that the pre-release failure rates are related to the failure rate experienced by end users. The models generally assume that failures occur once (after which the software is corrected and the cause of the failure is removed,) and that the failures are related to software faults and not configuration or usability issues. A survey of the reliability growth models can be found in [2, 4].

For a general purpose software product, once the product is released to a large number of users, the possibility of providing an accurate measure of reliability, shortly after product release, becomes more feasible as large amount of failure data can be captured from the end users through various data collection processes. For measuring reliability, a simple method that has been used in the past is to determine the total number of failures experience by all the operational units and then divide it by the number of units in the field. If we have N installations of the software, and a total of F failures are reported by all the installations in a time period T, then the failure rate of the software can be computed as $\lambda = F / (N * T)$ [9].

Using this approach to represent the failure rates of the product assumes that the failure rate depends only on the number of faults in the software, and independent of other factors such as the experience level of the people managing the system. In other words, it assumes that the failure rate is constant for a software product.

However, it has often been observed that for software products the failure rate decreases with time, even if no software changes are being made. For example, the

overall failure rates over time of an actual product as determined by the failures reported and the total number of units sold, is depicted in Figure 1. These are based on failure and population data of a real product; the actual data is given later in Table 1.

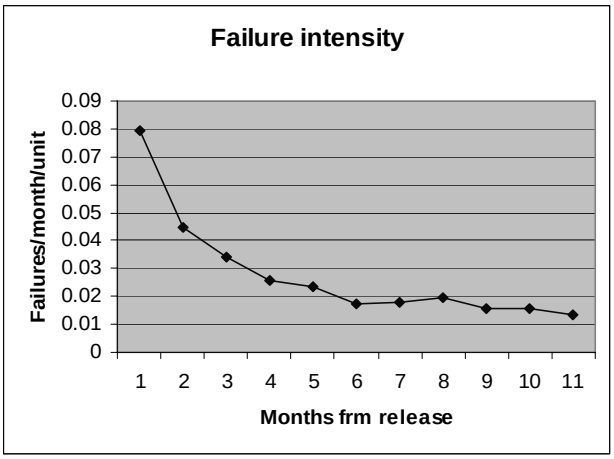

Figure 1: Overall failure rate of a product

This failure rate is measured across the total user base and during a period of time when there were no new versions of the product being released. As we can clearly see from this data, the failure rate for the product decreases with time. While this graph displays the average failure rate across all users, including the new as well as old users, the data clearly suggests that for a user the failure rate decreases with time.

In other words, for software products, we have the phenomenon that there is a growth in reliability without any fault removal. This phenomenon has been informally discussed by many, and other published data indicates this [1, 7]. Data from some Microsoft products also indicates the existence of this phenomenon (we will discuss data from some products later in the paper.) It should be clear that this reliability growth for products is different from the growth modeled in most of the software reliability growth models – it is due to large number of factors and not primarily due to fault removal. It is this reliability growth in the initial stages after product release which is the subject of this paper.

The immediate question is why the reliability of a software product improves with time for a user. One reason for this improvement is that the user learns to properly use the product and avoids the situations, commands, and actions that cause failure. By doing this, the user stops experiencing those failures that are repeatable, and is left to face only those that are random and unpredictable. That is, after initially experiencing a high failure rate, the user reaches a steady state failure rate for the software product. This reason has also been informally quoted by many people in product business. Another factor is configuration. Installing new software onto existing systems can result in versioning issues where this version of the software does not work with what is currently on the machine. Users will then upgrade drivers/applications to configure the new product such that it works properly.

This type of behavior of a product raises the question of what should be termed as the reliability of the product? The most natural choice is the steady state reliability of the product, especially as the failures that occur during this state are often the ones that are tested for prior to the product release. That is, we say that the reliability of a product is the steady state reliability of the average user achieved after the initial transient failures cease to occur. It should be clear that the average reliability, as measured by total failures and total population, does not represent the steady state reliability, as it combines the failure rates of new and existing users.

In this paper we propose a simple approach to model this phenomenon of reliability growth without fixes. We assume that the failure rate for a user decreases by a factor α every month, till it reaches the steady state. Using an initial failure rate, the factor α, and the steady state failure rate, if we know the number of total units of the product in the field every month, we can model the aggregate failures. Using the failures that are actually observed or reported, we can then determine the parameters, including the steady state reliability, using statistical techniques like the maximum likelihood principle.

In the next section we discuss the basic model of how failure rate changes with time. In section 3 we discuss how from using the failure rate model and data on monthly sales, the total number of expected failures can be determined. In section 4, we discuss how the model can be applied to a product, and in section 5 we give a few examples of using the model on data from actual products.

2. Failure Rate for a Unit

As mentioned previously, the failure rate experienced by a software unit of the product is high in the initial months following the release and then it reduces with time. To model the failure rate for a unit, we work with a month as the unit of time. For a product unit, we model that the failure rate experienced month i after it is purchased is given by

$$\lambda(i) = \lambda_0 * \alpha^i + \lambda_f$$

where $(\lambda_0 + \lambda_f)$ is the initial failure rate of the product on its release, α is the decay factor, and λ_f is the final steady state failure rate of the software product (which we refer

to as the product reliability.) That is, there is an *extra transient failure rate* of λ_0 when the product is initially installed. This transient failure rate decays with a factor α every month. After a few months, this transient failure rate approaches zero, and the user experiences the steady state failure rate of λ_f. In other words, the failure rate of a unit looks like as shown in Figure 2.

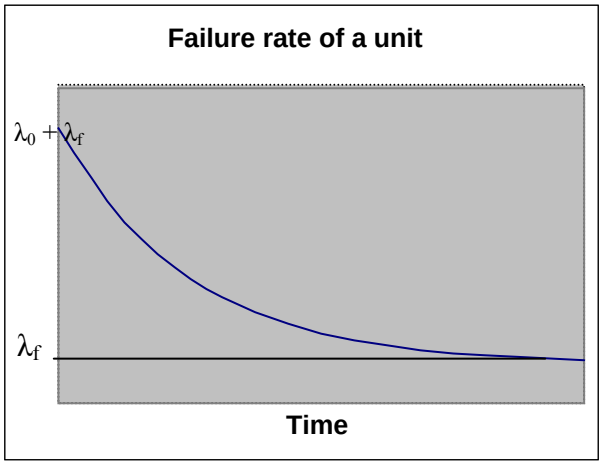

Figure 2. Failure rate of a unit

Note that this is the failure rate of an individual unit with respect to time. For a product, new units are continuously being sold. Therefore, at any point in time different units in the field are experiencing different failure rates depending on their "age", where age is the time from product installation. Therefore, the total failures experienced by a product are not simply a product of the failure rate of one unit and the number of units in operation.

3. Modeling Total Failures

The previous section described how the failure rate for a unit is a function of time. Therefore, in a given month i, if we know the units sold in all the months since the product was released till i, we can estimate the total number of failures that should be experienced in the field. This can be done since we know the number of units of different ages, and with the above model, we can determine the failure rates for units of different ages.

Suppose the total number of units sold each month, starting from the first month, is $N_1, N_2, N_3, \ldots, N_i$. Using the model for failure rate decay with time, we will get the following equations for the total failures in a month i, Fi, as:

$$F_1 = \lambda_0 N_1 + \lambda_f N_1$$
$$F_2 = \lambda_0 N_2 + \lambda_0 N_1 \alpha + \lambda_f (N_1 + N_2)$$
$$F_3 = \lambda_0 N_3 + \lambda_0 N_2 \alpha + \lambda_0 N_1 \alpha^2 + \lambda_f (N_1 + N_2 + N_3)$$
$$\ldots$$

By shifting the factors involving λ_f to the left hand side and substituting F_i' for $(F_i - \lambda_f (N_1 + N_2 + \ldots + N_i)$, we can rewrite the same equations as

$$F'_1 = \lambda_0 N_1$$
$$F'_2 = \lambda_0 N_2 + \lambda_0 N_1 \alpha = F_1' \alpha + \lambda_0 N_2$$
$$F'_3 = \lambda_0 N_3 + \lambda_0 N_2 \alpha + \lambda_0 N_1 \alpha^2 = F'_2 \alpha + \lambda_0 N_3$$
$$\ldots$$
$$F_i' = F'_{i-1} \alpha + \lambda_0 N_i$$

Replacing Fi' with $(F_i - \lambda_f (N_1 + N_2 + \ldots + N_i)$, we get

$$(F_i - \lambda_f (N_1 + N_2 + \ldots + N_i)) = (F_{i-1} - \lambda_f (N_1 + N_2 + \ldots + N_{i-1})) \alpha + \lambda_0 N_i$$

Simplifying this equation, we get

$$F_i = F_{i-1} \alpha + \lambda_0 Ni + \lambda_f ((N_1 + N_2 + \ldots + N_i) - \alpha (N_1 + N_2 + \ldots + N_{i-1}))$$
$$F_i = F_{i-1} \alpha + \lambda_0 N_i + \lambda_f (N_i + (1- \alpha)(N_1 + N_2 + \ldots + N_{i-1}))$$

Hence, using our failure rate decay model, we have this equation relating the total failures observed each month, monthly units sold, and the model parameters.

Through the failure data from the field each month, and the monthly sales information, it is possible to compute the parameters of the model, using statistical techniques. In other words, from the actual data on failures and number of units sold for many months, it is possible to determine the three model parameters – the steady state failure rate λ_f , the initial failure rate λ_0, and the decay factor α. This can be done using a statistical technique like the maximum likelihood approach. These three data values can be used by the product manufacturer to determine the product reliability (λ_f), its usability (λ_0), and the ability of the user base to "learn" about the product (α). The manufacturer can therefore address each of these, using a variety of techniques, to improve future products. Note this model assumes that no patches are being applied to the product; this factor is discussed later in this document.

4. Applying the Model to Products

The main problem in applying this model to real products is getting accurate data on the products failures and sales every month over a continuous period of time. Sales data is generally collected through the sales unit of the organization. In general it cannot be assumed that the units sold represent the units deployed, as products may be "sold" to distribution channels which may result in weeks or months between the "sale" and users actually

installing the product. While it is rarely possible to measure the actual rate at which the product is being deployed, it is generally possible to get the sales data. In this paper, for the purposes of modeling, we will assume that the product sale and deployment occurs within the same month.

The collection of failure data is problematic. (Some of the issues related to data collection and analysis are discussed in [6].) Most product organizations have some product support organization where users can call to report failures and issues. Through this product support organization (PSO), failures statistics are often collected (this data is also given to the product developers, who then fix the bugs for later releases of the product.) PSOs usually do not (and often cannot) distinguish between reports from new users and old users. They just report the issues, and provide the workaround to the users. PSOs are often the main source of failure data on products, and their data has been used in other reliability and availability computations [1, 5, 7, 10]. Automated approaches for recording certain types of failures also exist for server-type of systems [3, 8].

Though the failure data from PSO is often the best that is available, it is known that not all users report failures, and that a user does not report all the failures. Additionally only a certain class of failures will be reported and annoyances would, in general, be ignored. In other words, the failure data with PSO is not likely to be complete. Organizations often assume that a fixed percentage of failures are being reported (e.g. the analysis in [1] assumes that 10% failures are being reported through this mechanism.) We also assume that the percentage of failures being reported to PSO is a fixed percent of the total failures being experienced by all the users. Note that as long as the percentage of failures getting reported remains by-and-large the same, regardless of what percentage is used in the model, the trends and ratios will remain the same. The absolute numbers regarding failure rate will, of course, change with the percentage figure used.

In our examples, we use some fixed percentage, selected based on experience of people. Consequently, in the analysis the absolute numbers should be taken as indicative only, but the trends can be relied upon. To apply the model, the following data should be available:

Months after release	1	2	3	…	i
Monthly sales	N1	N2	N3		Ni
Total failures reported	F1	F2	F3		Fi

Unfortunately, even with the assumption that we know what percentage of failures is being reported, there are still issues with this data. Most product organizations use the data from the PSO to fix the defects which have caused the failures that customers are reporting. The "patches" to these defects are then made available to the customers. However, these patches typically do not update the product being sold (as it is not possible to change the CDs that are already in production and the distribution channel.) Often, patches are made available through update websites, and it is estimated that only a fraction of the users actually download the patches. (Usage of patches is very much dependent on the product and the process used for deploying patches.)

Periodically, all the patches are put together in a service pack (SP), which then forms the next version of the product, and is released like a product. That is, after an SP is released, it will be sent to the distribution chains resulting in customers generally getting the SP version of the product. Occasionally, a SP release of a product may also include new features, therefore as well as correcting known bugs it may also create a new category of bugs.

Clearly, if a SP has been released, then the total failure rate of a product is the failure rate of two different versions of the product, with the newer version presumably having a lower failure rate. Hence, to apply the model, we should not use the aggregate failure data of after a SP has been released, or should find some method of distinguishing failures of the two versions, which is often not possible (note the PSO will distinguish between major releases of a product but not minor releases i.e. the PSO will differentiate Windows 98 and Windows XP failures but not between Windows XP and Windows XP SP1). It is possible to combine failures of two different software versions, by having two sets of parameters. But that would complicate the equations as well as determining of parameters.

Assuming that we use only the data till the first SP has been released, we still have the issue of patches. The users who download patches will essentially be working with different software potentially having a different failure rate. In most setups, it is not possible to know how many users use the patches. The informal estimates are that less than 1% of the users bother to download the patches. One of the reasons is that patches sometimes solve the problem to which the user has already found a workaround. We assume that the impact of patches on the failure rate is minimal – that is too few users update their copy with the patches, and even if they do, their failure rates do not change substantially as due to the workaround they would have experienced the same failures. If this assumption does not hold, the impact of the patches will have to be estimated and the failure data suitably enhanced to remove that effect. A possible way of achieving this is to increase the value of α over time to represent the impact of patch installation.

5. Examples

Let us apply the above model to data for some products. In Table 1 we give the failure and sale data for a programming environment product, which we will refer to as Product A. The average failure rate shown earlier in Figure 1 is for this data. To preserve the confidentiality of the data, the data has been scaled. The months refer to the months after release.

Month	Total failures	Monthly Sales
1	367	4618
2	853	14385
3	835	5608
4	791	6186
5	956	9829
6	805	5584
7	967	8240
8	1218	7656
9	1031	4914
10	1144	5295
11	1058	7418

Table 1: Failure and sale data for a product

For this product (product A), no service pack has been released until the 11th month. Therefore, for this data, it is not necessary to consider the impact of the SP release on overall failure rates. Patches, of course, have been released, but as with other products, no data is available about what percentage of users have installed the patches. From this data, the total number of units and total failures is known and the total failure rate can be computed. This total average failure rate was shown earlier.

Now let us apply the model to this data. We determine the parameters using the method of least squares. In this approach, we numerically find the set of parameters such that the sum of squares of the predicted value of the total failures, and the actual value found in the field is minimized. Using this method, we get the following values: Initial transient failure rate, λ_0 as 0.04 failures/month, steady state failure rate, λ_f as 0.008 failures/month, and the decay factor α as 0.4.

From this, we can see that the steady state failure rate of product A is 0.008 failures per month, which is about one-sixth of the average failure rate that would be computed in normal method (i.e. by dividing the total failures by the total population) in month 2, and is about one-third of the average failure rate in month 4. Even as compared to failure rate of month 6, the steady state failure rate is about 50% of the average failure rate. This is the value that should be compared to any failure rates that may be predicted using the reliability growth models.

This example clearly shows how the average failure rate computation may give a very different view of the reliability of the product – the average is often much higher than the steady state reliability. The example also illustrates that the decay factor for the transient failure rate is quite high – within a month the initial transient failures reduce to 40%, and in two months are down to 16%. In other words, a user reaches close to the steady state failure rate within two to three months, and the first month is generally the worst.

Let us see how close the model represents the actual data. For this, we determine the failures predicted by the model, and then plot this data along with the actual failure data. The result is shown in Figure 3. As we can see, the model follows the actual data quite closely. The average error in the predicted value is less than 10%.

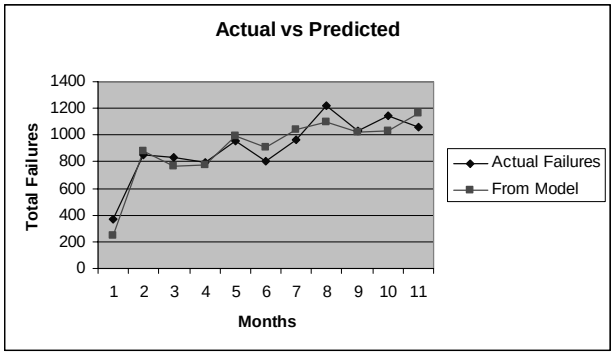

Figure 3: Total failures predicted and actual

We now apply this model to two other products, which we refer to as product B and product C. For these products also, the failure and sale data only until the time of release of the first service pack were used for applying the model. The value of the three parameters for the three parameters for all the three products is shown in Table 2.

	Product A	Product B	Product C
Initial transient failure rate, λ_0 (failures/month)	0.04	0.026	0.177
Steady state failure rate, λ_f (failures/month)	0.008	0.0066	0.067
Decay factor, α	0.4	0.24	0.10

Table 2: Model parameters for three different products

Both the products B and C are more complex than product A, with C being the most complex. Product B and C also require a lot more setup for proper operation as compared to product A, with product C requiring the most setup. We have not used any formal metrics for ranking the products for complexity, but have relied on internal information about the products and the relative team sizes of the products. Ranking for setup is also based on informal feedback from the support groups rather than any formal metrics.

The table shows that the decay of the transient failure rate is much sharper in product B, and is extremely steep in product C. This is consistent with what is to be expected – the more complex products that require more setup are likely to experience far more failures in the start. The average failure rate plots for these two products also show a very sharp decline in the average failure rate in the first month. In both of these, within a few weeks, a user reaches close to the steady state reliability of the product.

The model, however, has a larger error in these two cases. The average error in case of Product B product is about 16%, and in the case of Product C is about 29%. This suggests that when the transient failure rate decays rapidly, the accuracy of the model becomes lesser. A model where the decay factor alpha itself is a function of time and reduces with time may reflect reality better in these cases and may give a closer fit to the real data. However, that will make the model more complex, requiring a time dependent function for the decay factor. Of course, other functions are also possible to model this phenomenon.

6. Summary

It has often been observed that the failure rate for a software product often reduces with time, even without any changes being made to the product. One reason for this failure rate reduction is that with time users learn to avoid the situations, commands, and actions that cause failures. Another reason is that following installation users may encounter failures due to improper configurations of the system requiring the updating of the related components, drivers, applications, etc to work with the system. Most software reliability growth models do not model this phenomenon as they assume that reliability of a product depends on the number of faults in the software. Due to this phenomenon determining the steady state reliability of released products becomes harder.

In this paper we have proposed a simple model to represent this phenomenon. We propose that initially users of a product experience a transient failure rate, which decreases by a factor α every month. Eventually, this transient failure rate approaches zero and the users then experiences the steady state failure rate of the product, which we consider as representing the true reliability of the software product.

Using this model, from the total number of units being sold every month and the total number of failures being observed every month, it is possible to determine the three model parameters – the initial transient failure rate, the steady state failure rate, and the decay factor.

We have applied this model to failure and sale data for three different products. For these products, we found that the model closely represents the actual data, and the error between the predicted value of total failures and the actual value is quite reasonable. In our examples, for the simplest of the three products the decay factor was 0.4, while for the most complex it was 0.1, suggesting that the decay is more for more complex products that require more setup. The examples also show that the average failure rate (determined from total failures and total sales) can be substantially off from the true reliability, even after a few months.

In the examples, the error between the predicted failures and actual failures increases as the decay rate decreases. While it is less than 10% for the product with decay factor of 0.4, it is about 26% for the product with decay factor of 0.1. This suggests that the model of fixed decay every month might be suitable only for moderate decays. For more complex systems which are likely to experience larger transients in the start, perhaps the decay factor itself should be considered as a function of time.

In our examples, data for the first 8 to 12 months was used. However, it is desirable if predictions about steady state reliability and decay factor can be made early – this will help plan the support processes better. In this work we have not analyzed how accurate the predictions are when data from only the first few months is used, or how the accuracy changes as more data becomes available. We are currently examining some of these issues.

The model currently ignores the impact of patches, as no reliable data is generally available about the use of patches. We have also not attempted any correlation between in-process measures and the different parameters. These could be areas of further exploration.

It is likely that different types of products may have similar decay factors, even though the initial transient failure rate and the steady state failure rates will differ from vendor to vendor depending on the processes they use. It will be nice if eventually some categorization of the different decay factors for different types of products emerges. This, of course, will require a multi-organization study.

Acknowledgements. We would like to thank Mr. Vibhu S. Sharma, a Ph.D. student at IIT Kanpur, for helping with programs for parameter determination.

References

[1] R. Chillarege, S. Biyani, J. Rosenthal, "Measurement of failure rate in widely distributed software", Proc. 25th Fault Tolerant Computing Symposium, FTCS-25, 1995, pp. 424-433.

[2] W. Farr, "Software reliability modeling survey" in *Software Reliability Engineering,* Editor: M. R. Lyu, McGraw Hill and IEEE Computer Society Press, 1996, pp. 71-117.

[3] M. R. Garzia, "Assessing the reliability of windows servers", Proc. Conference on Dependable Systems and Networks (DSN), San Francisco, 2003.

[4] A. L. Goel, "Software reliability models: Assumptions, limitations, and applicability", *IEEE Transactions on Software Engineering,* Vol 11:12, 1985, pp. 1411-1423.

[5] J. Gray, "A census of Tandem system availability between 1985 and 1990", *IEEE Transactions on Reliability,* Vol 39:4, Oct 1990, pp. 409-418.

[6] W. D. Jones, M. A. Vouk, "Field data analysis" in *Software Reliability Engineering,* Editor: M. R. Lyu, McGraw Hill and IEEE Computer Society Press, 1996, pp 439-489.

[7] S. Kan, D. Manlove, B. Gintowt, "Measuring system availability – field performance and in-process metrics", ISSRE 2003, Supplementary Proceedings, pp. 189-199.

[8] B. Murphy, T. Gent, "Measuring system and software reliability using an automated data collection process", *Quality and Reliability Engineering International,* 1995.

[9] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications,* Second Edition, John Wiley and Sons, 2002.

[10] A. P. Wood, "Software Reliability from the customer view", IEEE Computer, August 2003, pp. 37-42.