

# On Effective Use of Reliability Models and Defect Data in Software Development

Rattikorn Hewett &  
Aniruddha Kulkarni  
Texas Tech University  
rattikorn.hewett@ttu.edu  
aniruddha.kulkarni@ttu.edu

Remzi Seker  
Dept of Computer Science  
University of Arkansas  
Little Rock  
rxseker@ualr.edu

Catherine Stringfellow  
Dept of Computer Science  
Midwestern State University  
Wichita Falls  
catherine.stringfellow@mwsu.edu

## Abstract

*In software technology today, several development methodologies such as extreme programming and open source development increasingly use feedback from customer testing. This makes the customer defect data become more readily available. This paper proposes an effective use of reliability models and defect data to help managers make software release decisions by applying a strategy for selecting a suitable reliability model, which best fits the customer defect data as testing progresses. We validate the proposed approach in an empirical study using a dataset of defect reports obtained from testing of three releases of a large medical system. The paper describes detailed results of our experiments and concludes with suggested guidelines on the usage of reliability models and defect data.*

## 1. Introduction

Software testing is an important process of software development to ensure the quality of software products. For large and complex software, testing becomes even more critical due to the lack of practical techniques for proving software correctness [4]. Although much research has advanced the techniques for generating test cases with high defect coverage, testing to guarantee defect-free software remains difficult if not impossible. Alternatively, an empirical approach to estimate remaining defects in software can help test managers assess the quality of software and make release decisions during testing. Various software reliability models (e.g., the delayed S-shaped model [9], the Musa model [5], the Yamada exponential model [10]) exist to predict the expected number of remaining or the total number of defects from previous defect data obtained during the testing process [5]. These models rely on various assumptions about software and its execution environment (e.g., defect repairs are immediate and perfect, testing effort never

changes, and must follow an operational profile, which dictates how software is likely to perform in the actual operations and environments).

Several issues on the applications of these reliability models arise in software development practices. First, the model assumptions do not always hold (e.g., imperfect repair, varying test effort due to economy or holidays, or testing that is based on system functionalities rather than operation profiles) [1]. Second, it is difficult to determine which model should be used although much empirical evidence has shown that these models are robust even when their assumptions are violated [7]. To make the matter worse, we cannot determine which model to use before testing begins. As testing begins, it is unlikely that any one model would be suitable across all testing stages and all releases. Finally, traditional defect analyses often focus on the defects discovered from the system testing (or system defect data) during development of a product release rather than the escaped defects discovered from the user/customer testing (or customer defect data). This is partly due to the fact that the customer defect data takes longer time to accumulate or may not be available. Consequently, software developers may trade timely delivery with high quality software products and settle with products that have acceptable remaining number of defects. While this practice works well in many software development projects, it may change in the future (or already now for certain applications e.g., safety-critical software) due to an increasing number of software intensive systems and rising demands on larger, more complex, and more dependable software.

One advantage of including customer defect data to predict the number of remaining defects in software is that, unlike the system defect data, the customer defect data follow the operational profile, a basic assumption in reliability models. While system testing can assist software verification, customer testing can help validate if the software product really performs in ways that the customer wants. In software technology today, several development methodologies (e.g., extreme programming

in agile computing, open source development) increasingly use feedback from customer testing and thus, customer defect data may become more readily available. Although much research has been done on software reliability models, most of it concentrates on the usage of system defect data and ignores the usage of customer defect data.

Our research aims to identify effective use of existing reliability models and defect data during testing in software development. In particular, we would like to have a better understanding of impact of model selection on the prediction of software defects from *both* types of defect data. To address some of the issues mentioned above, Stringfellow and Andrews [7] introduced a strategy for selecting appropriate reliability models as testing progresses. However, like most work in reliability modeling, they applied their approach to the system defect data. The strategy works well despite the fact that the testing at this stage is typically functional testing and thus violates the operational profile assumption for the model application. To effectively use the reliability models and defect data during software testing, we propose to additionally apply the strategy to the customer defect data, where testing at this stage generally follows the operational profiles. To investigate this idea, the goal of our empirical study is two-folded: (1) to evaluate the model selection strategy as to whether it provides consistent choice of models as selected when using only the system defect data, and (2) to see if the use of customer defect data gives better prediction of the expected number of remaining defects in software than the prediction obtained from system defect data. The study can provide guidelines on the usage of reliability models and defect data.

The rest of the paper is organized as follows. Section 2 discusses the details of the strategy for selecting reliability models as testing progresses. Section 3 describes the data set and experimental results on customer defect data. Section 4 compares modeling results obtained from using system defect data and that obtained from using customer defect data. The paper concludes in Section 5.

## 2. Reliability Model Selection

Because of the complexity of various interacting factors contributing to software reliability, it is unlikely that the “best model” exists cross the systems or in all releases. This section describes a selection method that determines the best model(s) for estimating the remaining number of defects in the software as presented in [7]. The strategy applies various reliability models to fit weekly accumulative defect data during system testing and estimate the expected remaining number of defects in software after release. The model is selected based on (1) how well it fits the data (measured by the goodness of fit

based on the  $R^2$ -value), (2) the prediction stability (a condition when the prediction in any week is within some small percentage of the prediction in its previous week), and (3) the difference between the actual and predicted numbers of defects. Figure 1 summarizes steps in the proposed strategy.

- 1) Apply models and estimate their parameters
- 2) If the model does not converge then reject the model
- 3) if  $R^2$ -value  $< \alpha$  then reject the model
- 4) if # of predicted defects  $<$  actual then reject the model
- 5) if the prediction is not stable then continue testing & go to 1)
- 6) the method prediction = the maximum # of predicted defects from all stable models
- 7) if the method prediction - actual  $<$  threshold then stop testing and software is acceptable for release

**Fig.1** Model selection strategy.

The application of reliability models in Step 1) is recommended to start after at least 60% of the testing effort is complete so that there is enough data to obtain stable models [5, 8]. Step 3) selects a model that fits the data well by checking if its goodness of fit is high enough. Steps 4) and 6) ensure conservative predictions to make sure that as many defects as possible are accounted for. Step 5) tests for model stability by checking if the predicted defect numbers of current week is within  $k\%$  of the predicted defect numbers of previous week. If it is, then the prediction is stable. Finally, Step 7) determines if the difference between the method prediction and actual number of defects is low enough to determine if the testing can terminate.

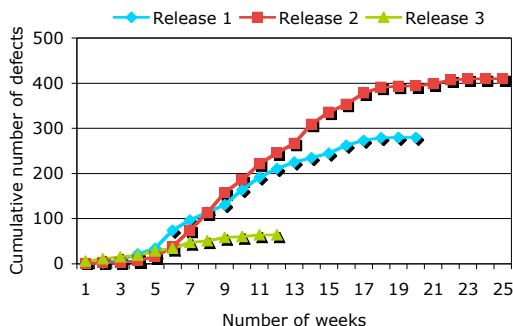
The strategy is general in that it can be applied to any set of reliability models, thresholds and statistical techniques for determining the degree of model fitting. However, the choice of the models and thresholds to be used can influence the results. More details of this strategy are described in [7].

## 3. Data and Experimentation

We use a defect data set obtained from testing of three releases of a large medical record system as reported in [7]. The system initially contained 173 software components and 15 components were added over the three releases.

Each row of the defect data set represents a defect report in each week with various corresponding attribute values including the defect report number, the release number, the phase in which the defect occurred (e.g., development, test, post-release), test site reporting the defect, the component and the report date. In this study we use the number of customer defects as reported in the test phase from the customer-testing group. Figure 2

shows cumulative number of these customer defects reported by week in all three releases. The defect data set also includes additional number of defects found in post-release of each release giving an *actual final number of defects* found to be 332, 443, and 70 for releases 1, 2 and 3, respectively.



**Fig.2** Customer defect data in all three releases.

Our experiments apply the strategy described in Section 2 using the same set of reliability models and threshold values as those employed in [7] for comparison purpose. In particular, the goodness of fit threshold  $\alpha$  is 95%, and the stability threshold  $k$  is 10% as suggested by Wood's study in [8]. We use four reliability models: the G-O model [2], the delayed S-shaped model [9], the Gompertz model [3] and the Yamada Exponential model [10]. Each of these models has at least two parameters representing a failure detection rate and a number of defects for infinite testing. The latter is referred to as an estimate or predicted defect numbers. We use NLREG [6], a commercialize tool for non-linear regression to estimate weekly predicted defect numbers.

#### 4. Experimental Results

Table 1 shows the results obtained from the customer defect data in Release 1. The application of the models starts after week 11, at approximately 60% of an overall testing plan, and the defects are reported weekly until week 20. Second column shows actual cumulative number of defects found each week. For each model applied, we list predicted cumulative number of defects with a corresponding  $R$ -value as used in [7]. We omit results from the Yamada model, whose prediction does not converge and thus, we reject the Yamada model by Step 2 of the model selection strategy. Since  $R$ -value of the G-O-Models is 0.92, which is less than the goodness of fit threshold of 95%, therefore by Step 3 of the strategy, we reject the G-O model in week 11. As shown in Table 1, both of the remaining two models: the Delayed S Shaped Model and the Gompertz Model are stable (as indicated in bolds, starting in weeks 16 and 12, respectively) and each has predicted number of defects higher than the actual

number of defects in the final week 20. Since the Delayed Shaped Model has higher number of predicted defects, by Step 6 of the strategy, we select the Delayed Shaped Model.

**Table 1.** Defect numbers for Release 1.

Test Week	Actual #Defects	GO-Model		Delayed S Shaped Model		Gompertz Model	
		Predicted #Defects	R-val	Predicted #Defects	R-val	Predicted #Defects	R-val
11	192	1417017	<b>.92</b>	3257	.99	275	.99
12	211	1481195	.92	1678	.99	<b>297</b>	.99
13	225	1309592	.92	1000	.99	296	.99
14	235	1742761	.92	699	.99	289	.99
15	245	1583544	.93	560	.99	286	.99
16	262	1937212	.94	<b>507</b>	.99	293	.99
17	273	1713836	.94	472	.99	299	.99
18	279	2225268	.95	442	.99	302	.99
19	280	1554480	.95	413	.99	301	.99
20	280	839583	.96	388	.99	298	.99

Similarly, Tables 2 and 3 show results obtained from the customer test data in Releases 2 and 3, respectively. In Table 2, the G-O model gets rejected due to low  $R$ -Value in week 13. The Gompertz model gets rejected because its predicted number of defects is less than the number of defects actually found in week 14. On the other hand, the Delayed S Shaped model becomes stable in week 22 and remains stable for the weeks ahead. It is the only stable model. Furthermore, it satisfies all selection criteria and so the strategy selects the Delayed S Shaped for Release 2.

**Table 2.** Defect numbers for Release 2.

Test Week	Actual #Defects	GO-Model		Delayed S Shaped Model		Gompertz Model	
		Predicted #Defects	R-val	Predicted #Defects	R-val	Predicted #Defects	R-val
13	267	1802098	<b>0.90</b>	4063352	0.99	319	0.99
14	309	1901480	0.91	24338	0.99	<b>286</b>	0.98
15	336	2287349	0.92	5998	0.99	294	0.96
16	353	2565727	0.93	2668	0.99	314	0.96
17	378	2688567	0.93	1814	0.99	403	0.99
18	391	2949379	0.94	1341	0.99	446	0.99
19	393	2612969	0.95	1030	0.99	442	0.99
20	394	2526553	0.96	841	0.99	434	0.99
21	398	3073040	0.96	729	0.99	429	0.99
22	407	2062242	0.96	<b>664</b>	0.98	428	0.99
23	409	2410185	0.96	617	0.98	426	0.99
24	410	1235062	0.96	581	0.98	425	0.99
25	410	7603	0.96	553	0.98	423	0.99

**Table 3.** Defect numbers for Release 3.

Test Week	Actual #Defects	GO-Model		Delayed S Shaped Model		Gompertz Model	
		Predicted #Defects	R-val	Predicted #Defects	R-val	Predicted #Defects	R-val
7	47	304093	0.97	103	0.95	3778	0.99
8	51	372814	0.98	<b>100</b>	0.97	148	0.98
9	58	431444	0.98	103	0.98	113	0.99
10	60	380895	0.98	92	0.98	88	0.99
11	64	248079	0.98	88	0.99	<b>82</b>	0.99
12	64	488	0.98	82	0.99	76	0.99

For results of Release 3 in Table 3, the G-O model has acceptable  $R$ -value but it never becomes stable. Moreover, it exceedingly overestimates the defect numbers. Both the Delayed S Shaped model and the Gompertz model become stable respectively, at week 8 and 10, and in the remaining weeks. However, the Delayed S Shaped model is selected as it gives higher predicted number of defects of the two.

Testing usually continues for a few more weeks after each release and additional defects could be found. To assess the quality of our results in real practices, we need to take additional number of defects found in post-release into account. Table 4 shows the prediction obtained from all stable models in each release and compares them with the actual final number of defects that includes additional defects found during post-release. The *error difference* is computed by subtracting the actual final number of defects from predicted defects at the final test week. The *relative error* is a ratio between the error difference and the predicted number of defects at the final week in percentage.

**Table 4.** Validation with post-release defects.

Stable Models	Actual #Defects at final test wk	Actual final #Defects	Predicted #Defects at final test wk	Error Diff.	Rel. Error	Actual #Defects in post-release	Predicted #Defects in post-release
Release 1							
Del.S Sh.	280	332	388	+56	0.144	52	108
Gompertz	280	332	298	-34	-0.114	52	
Release 2							
Del.S Sh.	410	443	553	+110	0.248	33	143
Release 3							
Del.S Sh.	64	70	82	+12	0.146	6	18
Gompertz	64	70	76	+6	0.079	6	

As shown in Table 4, in Release 1, the Gompertz model under estimates by 34 defects whereas the Delayed S Shaped model over estimates by 56 defects. Though relative error is smaller for the Gompertz model, in practice, the Delayed S Shaped model is considered to perform better because it predicts a higher number of defects than the Gompertz model. In general, models that estimate slightly more are better than those that estimate less because the cost of fixing post-release defects is very high compared to fixing them before release. Thus, the selection of the Delayed S Shaped model in practice agrees with the results obtained by the strategy.

In Release 2 the Delayed S Shaped model is the only stable model to be considered. Its corresponding error difference and relative error appear to be acceptable (judged by experts) for the model to be applied. This is consistent with our result obtained from the model selection strategy. Similarly, in Release 3 both the Gompertz model and the Delayed S Shaped model over estimate slightly but we choose the latter since it is more conservative.

The last two columns of Table 4 compare the actual number of post-release defects and those predicted by the selected model. The number of remaining defects here is the defects found by all groups after the final week and it is to be expected since at least some defects will be found in later operation of the software. In all releases the predicted numbers of the remaining defects are higher than the actual ones. This is desirable in software practice since the cost incurred from fixing post-release software can be prevented.

## 5. System vs. Customer Defect Data

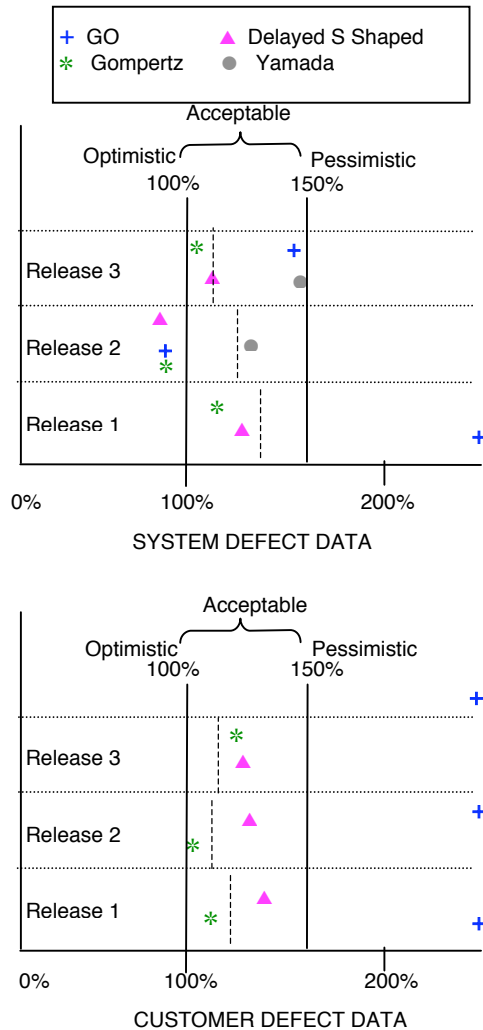
To understand the implications of the usage of system defect data, we compare the results obtained from customer defect data with those obtained from system defect data as reported in [7].

Table 5 summarizes the results of the model selection strategy applied to both types of defect data. The table contains all stable models, one of which is selected (as shown in bold). The results indicate that the strategy performs consistently for Releases 1 and 3. However, in Release 2 the result is very different. The usage of system defect data suggests applying the Yamada model that does not even converge in the case of customer defect data. However, the Delayed S Shaped model from customer test defect data gives  $R$ -value of 0.98, which is slightly higher than that of 0.97 from the Yamada model using the system test defect data [7].

**Table 5.** Results from model selection.

Stable Models	System Test Defect Data	Customer Test Defect Data
Release 1	<b>Delayed S Shaped</b> Gompertz	<b>Delayed S Shaped</b> Gompertz
Release 2	<b>Yamada</b>	<b>Delayed S Shaped</b>
Release 3	<b>Delayed S Shaped</b> Gompertz	<b>Delayed S Shaped &amp;</b> Gompertz

Figure 3 visualizes the results in more details by using  $n$ , the actual number of defects found at the final week as a base for comparison. The horizontal axis represents a ratio between a number of defects and  $n$  in percentage. Thus, results at 100% signify results at the final test week and all other results are compared relative to the actual number of defects found at the final week. The vertical dashed line shows a ratio of the actual final (total) number of defects to  $n$  in percentage for each release. For each release, the percentage of a ratio of a predicted number of defects to  $n$  is shown by a symbol that corresponds to the model employed except for non-converging models. The symbols appear on the far right represent very high percentages that may not be properly scaled. In general, the horizontal distance between each model symbol to the vertical dashed line represents its relative error defined in previous section.



**Fig. 3** Relative errors from both types of defect data.

The top part of Figure 3 shows the results obtained from using system defect data as reported in [], whereas the bottom part shows the results obtained from using the customer defect data. If we define an acceptable range of defect ratios to be between 100% and 150%, then all stable models including the model selected by the strategy from both types of defect data are acceptable. A model, which predicts number of defects less than that actually found (in which case it is rejected) or a number too close to the number of defects already found can be viewed as optimistic. The other way round we can view a model, which makes too high a prediction as pessimistic. In practice a pessimistic view within an acceptable range is preferred as fixing defects in post-release is more costly than in pre-release. However, the determination of an acceptable range may require a subjective opinion from an expert or a manager.

As shown in Figure 3, each selected model in all releases from customer defect data has a positive relative error (i.e., they overestimate the actual final defect number and thus, are on the right of the dashed line). Furthermore, they are pessimistic and within acceptable ranges of even as low as 100% to 130%. On the other hand, the selected models from system defect data underestimate the actual final defect number with the exception of Release 2. However, in Release 2 the relative error of the Delayed S shaped, which is the selected model in customer defect data case is more pessimistic than the relative error of the Yamada model selected for system defect data case.

## 6. Conclusion

Our results show that the model selection strategy performs consistently for both types of defect data. The same reliability models are selected for prediction in all but second software release. In addition, the gap between the prediction and the actual remaining number of defects obtained by using the customer defect data is more consistent to the expected estimate in real development practices in the sense that it supports pessimistic view within an acceptable range. Thus, the experiments validate that using customer defect data can provide predictions of remaining number of defects that are inline with practices. This suggests that when customer defect data are available they should be incorporated into the application of reliability models and the model selection strategy to provide effective use of defect data to realistic defect prediction that can assist software release decisions.

## References

- [1] Fenton, N. E. and M. Neil, 1999. A critique of software defect prediction models, *IEEE Trans. Software Engineering*, 25(5): 675–689.
- [2] Goel, A. and K. Okumoto, 1979. A Time Dependent Error Detection Model for Software Reliability and Other Performance Measures, *IEEE Transaction on Reliability*, 28(3): 206–211.
- [3] Kececioglu, D., 1991. *Reliability Engineering Handbook, Vol. 2*, Prentice-Hall, Englewood Cliffs, N.J.
- [4] Littlewood, B. and L. Strigini, 1993. Validation of ultra high dependability for software-based systems, *CACM*, 36(11): 69–80.
- [5] Musa, J. D., A. Iannino and K. Okumoto, 1987. *Software reliability: measurement, prediction, application*, McGraw-Hill, Inc., New York, NY, USA.
- [6] P. H. Sherrod, 1998. *NLREG 4.0: a non-linear regression algorithm shareware*, <http://www.nlreg.com/index.htm>.
- [7] Stringfellow, C. and A. Andrews, 2002. An empirical method for selecting software reliability growth

models, *Empirical Software Engineering*, 7(4): 319–343.

- [8] Wood, A., 1996. Predicting software reliability, *IEEE Computer* 29 (11):69-78.
- [9] Yamada, S., M. Ohba and S.Osaki, 1983. S-Shaped Reliability Growth Modeling for Software Error Detection, *IEEE Transaction on Reliability*, 32(5): 475–478.
- [10] Yamada, S., H. Ohtera and H. Narihisa, 1986. Software Reliability Growth Models with Testing Effort, *IEEE Transaction on Reliability*, 35(1): 19–23.