Software Defect Prediction Based on Stability Test Data

Kazu Okumoto
Alcatel-Lucent
Naperville, IL, USA
kazu.okumoto@alcatel-lucent.com

***Abstract*—Software defect prediction is an essential part of evaluating product readiness in terms of software quality prior to the software delivery. As a new software load with new features and bug fixes becomes available, stability tests are performed typically with a call load generator in a full configuration environment. Defect data from the stability test provides most accurate information required for the software quality assessment. This paper presents a software defect prediction model using defect data from stability test. We demonstrate that test run duration in hours is a better measure than calendar time in days for predicting the number of defects in a software release. An exponential reliability growth model is applied to the defect data with respect to test run duration. We then address how to identify whether estimates of the model parameters are stable enough for assuring the prediction accuracy.**

Keywords-software defect data; software defect prediction; stability test; test duration; exponential reliability growth model

I. INTRODUCTION

The following terms are used interchangeably here for simplicity: outages vs. failures and defects vs. faults vs. errors. A software failure is defined as a system outage caused by a software defect. Software reliability is defined as rate of software failures or outages. Software defect predictions typically assume that there are a finite number of defects that can be exposed in a particular operational profile. Thus, as residual defects are discovered and removed, there are fewer defects left to be exposed. Fewer critical residual software defects should be encountered less frequently in normal operation, and thus should yield a lower software failure rate. It should be noted that not all defects will result in field outages or failures. In this paper we focus on defect predictions. A software defect model is a key step for addressing customer-perceived reliability measures [5].

There are typically two types of software reliability testing (robustness and stability tests) to be performed to ensure highly reliable software prior to the software delivery. Results of robustness and stability tests should contribute to the validation of reliability requirements and the predictions of system availability and reliability. Robustness testing (also called 'negative testing', 'adversarial testing', or 'fault insertion testing') confronts systems with plausible failure scenarios to assure that automatic failure detection, isolation, and recovery mechanisms work rapidly and reliably.

Stability (or 'endurance') testing uses a heavy, mixed traffic load against a system for an extended period, typically at least 72 hours. The stability run is performed to track whether new software features and code have impacted the overall network/solutions stability. Stability tests should be performed on final software release and after all feature testing is complete. This will assure that testing is against final product. A best practice is to simulate both heavy end user and Operation, Administration, Maintenance & Provisioning (OAM&P) activity. It is important to select the operational profile to replicate the actual operating environment in testing. Quantitative estimation of parameters is only meaningful if the operational profile during testing is representative of the field environment.

In this paper we will demonstrate that an exponential reliability growth model based on stability test run duration provides accurate predictions for residual defects which will significantly impact system stability in the field.

II. DEFECT DATA

The defect data should be normalized against test effort (e.g., stability run duration hours, tester-days) rather than calendar time, as pointed out first by Musa [3]. It will eliminate non-uniform effort over testing interval, variations in staffing levels, weekends, holidays, etc.

Typical software defect data is shown in Fig. 1 with stability run duration in hours and calendar time in days. The data was taken from a relatively large-scale software development with over 2 million lines of code for a next generation radio network control system. The defect data and run hours were collected on a weekly basis. We can observe the defect find rate continuously leveling off with stability run hours as expected. This is referred to as software reliability growth through the find-fix process. As residual defects are discovered and removed, there are fewer defects left to be exposed. However, continuously leveling off behavior is not obvious for the calendar time based defect find rate. It will be shown in Section 3 that the defect data based on stability run duration hours can be well represented by an exponential model.

Further examination of the test effort data is illustrated in Fig. 2, where the test execution rate is low at the startup period. This is mainly due to system stability problems or tools/lab environment problems. Once the early critical issues were removed, the execution rate became constant. Toward the end of the stability test the effect is diminishing, i.e., the defect find rate is sufficiently low, indicating the readiness of the software delivery. In fact, the project decided to move most of the test resources to the next release.

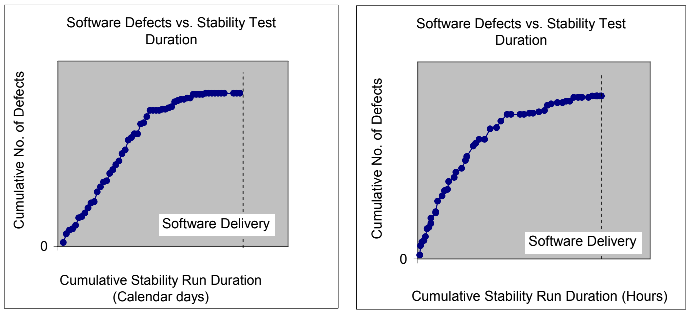

Figure 1. Defect data based on calendar time vs. test run duration hours

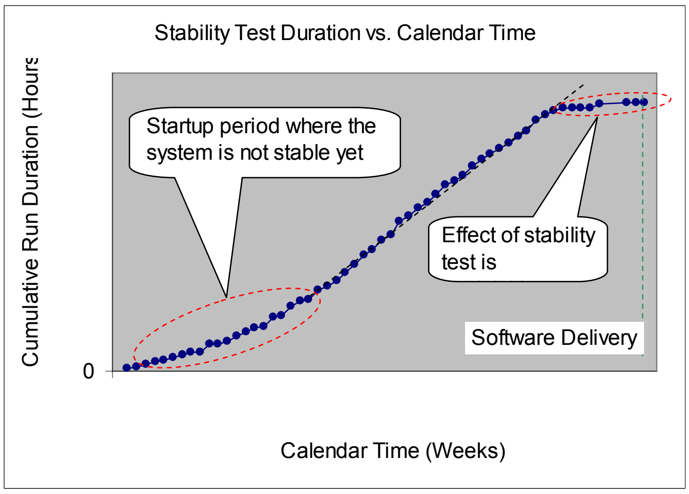

Figure 2. Stability run duration hours vs. calendar time

In the following sections we will quantify the above observations using an exponential reliability growth model.

III. SOFTWARE DEFECT PREDICTIONS

Software defect prediction models are known as software reliability growth models. A large number of prediction models have been proposed and investigated over the last four decades. There are many references documenting and comparing various models in detail (e.g., Musa-Iannino-Okumoto [4], Lyu [2], Wallace-Coleman [6]).

Goel-Okumoto[1] first formulated this defect find process as a stochastic process in terms of defect find interval or time-varying defect find rate. Since we count defects as they are exposed, it seems logical to statistically formulate the number of defects found during test as a Poisson process with a time-varying mean value function, which is known as a non-homogeneous Poisson Process (NHPP). That is, for a defect find process, N(t), the probability of finding n defects by time t is expressed as a Poisson distribution with the mean value function, m(t), as:

$$P\{N(t) = n\} = m(t)^n \exp\{-m(t)\} \,/\, n! \qquad (1)$$

It is typically assumed that there are a finite number of defects in any piece of software. Most of those frequently used models can be systematically sorted in terms of the shape of the mean value functional (i.e., an exponential curve or an S-shaped curve).

In this paper we use an exponential model, which is simple and proved to provide predictions as accurate as, if not more than, S-shaped models. This will be also confirmed in the following analysis.

An exponential model is usually represented as an NHPP with the mean value function:

$$m(t) = a\,\{1 - \exp(-b * t)\}, \qquad (2)$$

where $m(t)$ = cumulative number of defects found at time t, a = total defects in the software, and b = rate at which each defect is exposed or found.

The corresponding defect intensity function or defect rate can be derived as the derivative of the mean value function:

$$\lambda(t) = a\,b \exp(-b * t). \qquad (3)$$

The maximum likelihood method is commonly used for estimating the parameters, a and b, for a give set of defect data. The specific estimation procedures are described in [4]. Using the stability run duration data shown in Fig. 1, we have obtained the maximum likelihood estimates for a and b, substituted them in the exponential function (2), and overlaid the predicted curve with the actual data, as illustrated in Fig. 3. It demonstrates a remarkably good representation of the defect data. Using the Poisson distribution, we also provided the 90% lower and upper limits. It helps validate that actual defect data follows a Poisson process with the exponential mean value function.

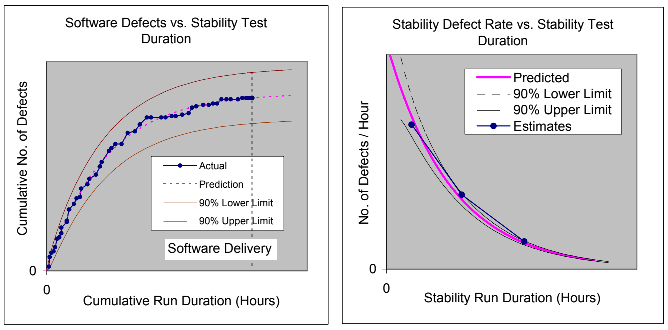

Figure 3. Stability defect data with the defect prediction model

Similarly, substituting the maximum likelihood estimates for a and b into (3), we can obtain the predicted defect rate. The 90% limits can be easily converted from the cumulative curves. The defect intensity was estimated based on the actual defect data. Each data interval contained sufficiently large number of defects to avoid possible small sample size problems. The estimated defect intensity data are shown in Fig. 3, along with the predicted defect rate with the 90% limits. It demonstrates that the actual defect rate point estimates are within the 90% limits.

IV. Determination of Model Parameters Stability

Having demonstrated the validity of the exponential model, we will now address how to determine whether the estimates are stable enough to be used for predictions. Estimates of the model parameters are typically updated as a new set of defect data becomes available. For this data set we monitored the changes in the parameter estimates (a and b) for every new data point, as shown in Fig. 4.

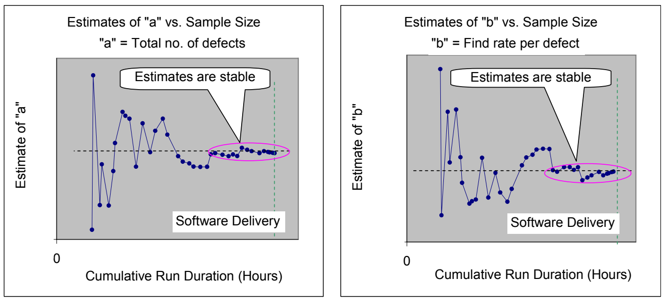

Figure 4. Maximum likelihood estimates of a and b with sample size

In order to illustrate a method to determine the stability of the parameter estimates, we used the parameter, *a*, and calculated the median of the estimates from the beginning to the latest data point. The median was used to reduce the influence by possible outliers. Fig. 5 shows the median over the sample size, overlaid with actual estimates. It can be easily seen that the median of the estimates became stable when actual estimates started stable.

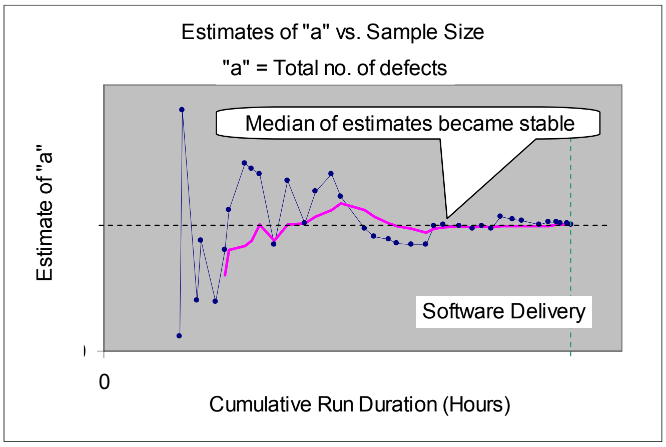

Figure 5. Median of estimates of a and actual estimates with sample size

We used the estimates at the time when the median became stable to illustrate the predictions compared against future defect data, where the conditional limits were used. Assuming that we have found n_e defects after t_e hours of stability runs, the conditional probability of finding n defects by t ($>t_e$)hours of stability runs can be expressed as:

$$P\{N(t) = n \mid N(t_e) = n_e\} = \{m(t) - m(t_e)\}^n \exp[-\{m(t) - m(t_e)\}] / (n - n_e)! \quad (4)$$

As shown in Fig. 6, actual defect data after te hours of stability runs continues to be within the limits, that is, in line with the predictions.

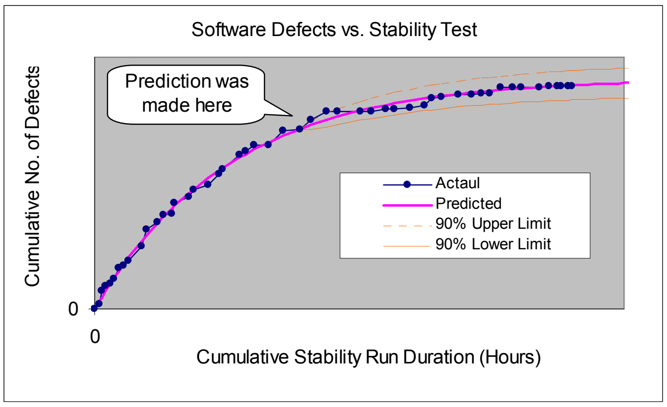

Figure 6. Software defect predictions at the time when the estimates are stable

V. Conclusions

We have demonstrated that stability test duration in hours is a better measure than calendar time in predicting the number of defects in a software release. An exponential reliability growth model based on stability test duration provides accurate predictions for the residual defects at the end of stability test. We also addressed how to identify whether estimates of the model parameters are stable enough for assuring the prediction accuracy.

Reference

[1] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," IEEE Transactions on Reliability, pp. 206-211, 1979.

[2] M. R. Lyu. Handbook of Software Reliability Engineering. New York: Computer Society Press, 1995.

[3] J. D. Musa, "Operational Profiles in software-reliability engineering," IEEE Software, pp. 14-32, 1993.

[4] J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987.

[5] K. Okumoto, "Customer-perceived software reliability predictions: Beyond defect prediction models," Stochastic Reliability and Maintenance Modeling, 2011.

[6] D. Wallace and C. Coleman, "Application and improvement of software reliability models," Hardware and Software Reliability, pp. 323-08, 2001.