Early Software Defect Prediction: Right-Shifting Software Effort Data into a Defect Curve

Kazuhira Okumoto
Sakura Software Solutions LLC
Naperville, IL USA
kokumoto@sakurasoftsolutions.com

***Abstract*—Predicting the number of defects in software at release is a critical need for quality managers to evaluate the readiness to deliver high-quality software. Even though this is a well-studied subject, it continues to be challenging in large-scale projects. This is particularly so during early stages of the development process when no defect data is available. This paper proposes a novel approach for defect prediction in early stages of development. It utilises a software development and testing plan, and also learns from previous releases of the same project to predict defects. By producing key quality metrics such as percentage residual defects and percentage open defects at delivery, we enable decisions regarding the readiness of a software product for delivery. Over several years, the approach has been successfully applied to large-scale software products, which has helped to evaluate the stability and accuracy of defects predicted at delivery over time.**

***Index Terms*—Early defect prediction, Software quality assurance, Software reliability growth modeling, Software effort data**

I. Introduction

Project managers want to balance development resource allocation and software quality at delivery. It is critical to understand where software defects[1] are injected and removed during the software development process to evaluate software quality. Figure 1 summarizes defects injected and removed during the software lifecycle. The process begins with content (or features) specification, followed by software design, coding & unit test, and formal testing (such as integration test, feature test, and system test). As the internal test is winding down, the software is typically delivered to the customer site for an acceptance test or joint customer site test, followed by the deployment for customer operation. Defects are introduced during requirement specification and code development (i.e., design, coding, and unit test). Defects are found and removed during the entire lifecycle. However, defects found by independent testers and customers are typically reported and available for analysis. A subset of defects during the customer operation often causes a system outage (or failure), resulting in a full or partial system downtime. These defects are related to the software reliability and availability. However, the terms are often used interchangeably with other types of defects in the literature.

[1]In this paper, we use the terms such as defects, faults, and bugs interchangeably.

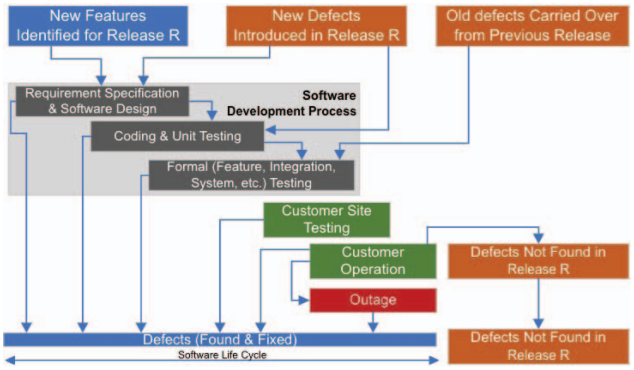

Fig. 1: Software Development Process vs Defect Injection and Removal. Defect = Faults = Bugs. Outages = Failures.

In this paper, we introduce an innovative method for automatically generating multiple curves for defect prediction. It provides consistency and accuracy for defect prediction during the test phase. We offer in-depth technical discussion. It is entirely different from a traditional single curve representation of the defect trend.

II. Related Work

Methods for predicting software defects over time are often called software reliability growth models using defect trend analysis. It looks at the cumulative defect detection (or arrival) curve to determine the shape of the curve. Over 220 models and a few tools (e.g., [1]–[7]) have been established since the early 1970s, but how to put them in practice remains unsolved primarily. This is mainly because defect data from actual projects are not readily available to researchers in the field. It started with a simple exponential curve [7] for defects found or detected during a system test phase over 40 years ago. The assumption is that a software product is developed and stable. There is a finite number of defects in the software, and each defect is detected according to an exponential distribution or a constant rate. The number of defects detected can be formulated as a non-homogeneous Poisson process (NHPP). It is simple enough to develop a statistical method for estimating parameters using the maximum likelihood method [7], [8]. It represents the defect trend during system tests because the software content is mainly developed and reasonably stable.

978-1-6654-7679-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ISSREW55968.2022.00037

Since then, it's been extended to cover the entire test phase, including feature, integration, functional, and system tests. The original exponential model has been modified to be flexible, e.g., substituting a different distribution function into the exponential distribution function. It accommodates various shapes of defect data. But it becomes complex. They are often called S-shaped models [1], [2], e.g., Weibull, Gamma, and logistic. However, none of the models have a statistical method of estimating parameters due to the complex nature of the curve. Curves are usually generated manually by subject matter experts. It turns out that most S-shaped models cannot explain an early part of defect data if focused on the latter part of the defect data. A single curve approach does not appear sufficient to describe the entire defect data.

Recently we have developed an innovative method [9], [10] for defect trend analysis. It uses multiple curves instead of a single curve to describe the entire defect trend over time. It improved prediction accuracy and stability over the traditional models—no more single curve fitting. We have developed an innovative algorithm to automatically identify the trend changes or inflection points to generate various curves. It's now practical in use due to the advancement of computing technology. We will briefly describe the method in Section III.

We have developed another innovative method [11] for a project manager to balance development resources vs. software quality at the delivery date. It provides defect arrival (or detection) curves without actual defect data to be used for an early phase of the project, and hence, it is called an early defect prediction. It is described in Sections IV and V with or without actual defect data, respectively. Stability and accuracy of prediction will be addressed in Section VI.

III. Defect Trend Analysis

This section addresses our new innovative method for defect trend analysis. The earlier version of the technique was introduced in [9], [10]. Since then, we have significantly improved the algorithm. There are two parts: multiple curve generation and adjustment for the last curve.

A. Inflection points for generating multiple piece-wise exponential curves

The first step is to find inflection points or points in time where the defect trend changes significantly. Figure 2 shows a typical defect trend over time. There are multiple periods where the trend significantly changes from period to period. We apply a simple exponential curve as new defect data is added, keep track of the closeness between the predicted curve and actual data, and identify the significant change in the closeness. Maximum likelihood estimates for a and b are computed for every data point added. Once the inflection point is found, we move the start date to a new start date corresponding to the inflection point. Repeating these steps until the end of the defect data, we will find the final predicted curve, as shown in Figure 3. The algorithm in Fig. 4 provides a high-level procedure for finding inflection points and multiple

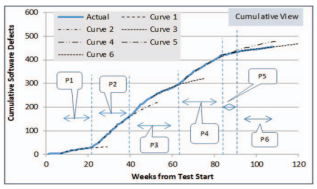

Fig. 2: Typical defect trend curve with multiple periods of various sub-trends

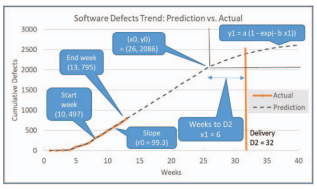

Fig. 3: Predicted Curve and Illustration of the last curve modification

curves. It is a series of piece-wise exponential models with a general form of:

$$m_i(t_i) = a_i(1-\exp(-b_i t_i)) + m_{i-1}(t_{i-1}) \quad (1)$$

where $m_i(t_i)$ is the cumulative defect data at time or week t_i for period i. Parameters a_i and b_i parameters represent total defects and detection rate for period i. The multiple curves represent a software development process where each subset of software modules is added to the test suite as it becomes available for testing. It creates various waves of defects being detected during the test phase over time. A single curve approach will not be able to accomplish the defect trend. The advancement of computing technology makes complex computing possible.

B. Adjustment for the last curve

Most models for trend analysis require a relatively large number of data points, i.e., weeks and several weeks closer to the delivery date. In other words, the data trend must be leveled off for the models to be effective, as illustrated in Figure 3. In practice, there are some cases (especially during an early test phase) where the last curve looks like a straight line. This creates a problem since we are expecting a finite number of defects. The latest curve of the multi-curve method needs adjustment to start leveling off several weeks before delivery. The adjustment is to extend the straight line to several weeks before the delivery date and generate an exponential curve that meets a target of 15% residual defects at the delivery date, as highlighted in Figure 3. In other words, the initial arrival rate at r_0 is the slope:

$$a \times b = r_0 \quad (2)$$

Note that we have

$$\%\ \text{Residual Defects} = \frac{\text{Total Defects} - \text{Defects at } D_2}{\text{Total Defects}} \quad (3)$$

Values for a and b can be determined by solving eq. 2 and 3. Past historical project data can specify values for weeks to the delivery date, and % residual defects.

IV. Early Defect Prediction Without Actual Defect Data

Defect trend analysis is relatively simple and easy to use in practice. However, it has some limitations. It requires defect data several weeks before the delivery date for the prediction to become stable. It is not appropriate for defect prediction during the planning phase where actual defect data is not available,

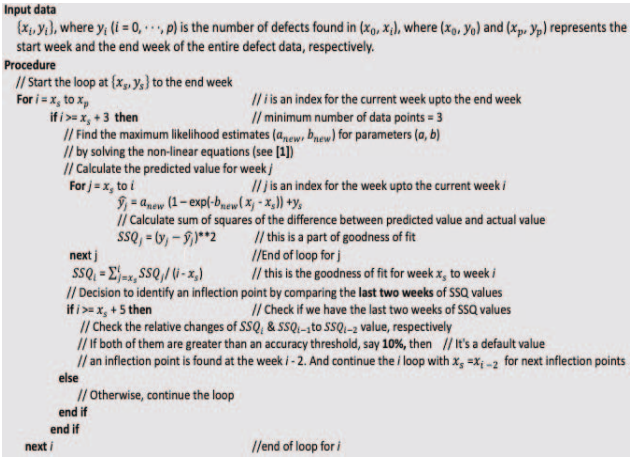

Fig. 4: A procedure for finding inflection points and multiple curves

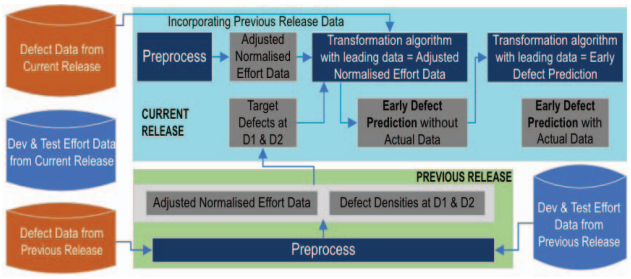

Fig. 5: Adjusted normalized effort curve vs. defect arrival curve

called an early defect prediction. We will introduce a concept of leading data to assist in early defect prediction. We have investigated possible leading data last several years based on the availability and the correlation with defects and concluded effort data is best suited for this purpose.

This section describes specific steps for early defect prediction, including closure and open curve predictions. We also validate the predicted curves by overlaying actual data. An overview of the process is illustrated in Figure 5. It incorporates previous release data using effort data as leading data.

A. Input data

Project milestone dates are quired. The start date, D_0, represents the start of testing. We expect to see defect data coming in shortly after D_0. A project typically has two delivery dates: Delivery for trial (D_1) and Delivery for commercial deployment (D_2). In addition, we need two sets of data from the previous release, i.e., defect data and effort data. Defect data is typically sorted every week in a cumulated format. Two types of effort data (development and test) are readily available during the planning phase in practice. It is usually measured in hours of effort required for completing a development activity, typically at a sub-feature level.

- Development effort data: represents the complexity of the software content. The development effort curve represents how the content is developed over time. It is used to predict the number of defects.
- Test effort data: represents the test progress if cumulative test effort is normalized. The defect detection or find rate is highly related to the test progress. In other words, the shape of the defect find curve is closely related to the test progress curve.

Typical cumulative development and test effort data are shown in Figure 6. Note that there is a significant gap between the development and test effort curves. For a large-scale development, many low-level modules need to be integrated for a feature to be ready for the test. The defects are reported from feature-level tests.

B. Preprocessing

We first prepare effort data suitable for the prediction algorithm by combining development and test effort to generate the leading data.

1) Normalizing development effort data by the test progress: We first normalize development effort data by the test progress, which is the test effort curve divided by the maximum value of the test effort data. It is calculated as:

$$\%\text{Test progress} = \frac{\text{Test effort data}}{\text{Max of test effort data}} \tag{4}$$

$$\text{NDE} = max(\text{DE}) \times \%\text{Test progress} \tag{5}$$

Where DE is the development effort and NDE is the normalised development effort. The normalized development effort curve represents how the software content is tested over time. It is shown in Figure 6.

2) Adjusting the tail end of the normalized development effort curve: Next, we will have to adjust the tail end of the normalized curve since it shows a rapidly flattening trend. This is because test effort represents internal test activities. Defect data usually includes those found by customer testing and operation. The adjustment will compensate for customer defects. The adjustment can be made using the trend analysis algorithm discussed in Section III.

3) Calculation of defect density: We are now ready to calculate defect densities at D_1 and D_2 using the defect curve and the adjusted normalized effort curve. An interpolation method is used for identifying values at D_1 and D_2, respectively. We can then calculate defect density DD as:

$$\text{DD} = \frac{\text{Defects}}{\text{Effort Hours}} \tag{6}$$

Figure 7 shows the adjusted normalized effort curve with overlaid defect arrival curve. As mentioned earlier, the defect curve is highly related to the effort curve.

4) Target defect values at the delivery date: We perform steps described above for the effort data from the current release to derive the adjusted normal development effort curve. Using the defect densities, DD, obtained in 6, we can compute target defects (TD) at D_1 and D_2 for the current release as:

$$\text{TD} = \text{Effort Hours} \times \text{DD} \tag{7}$$

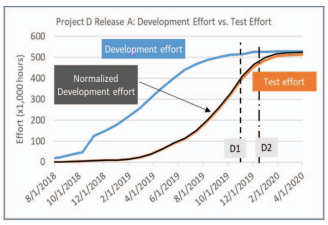

Fig. 6: Effort data: Development, test, normalized development

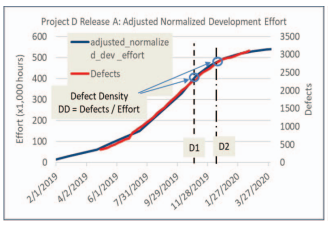

Fig. 7: Normalized development effort curve with the adjustment

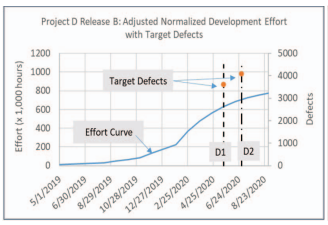

Fig. 8: Target defects vs. adjusted normalized development effort curve

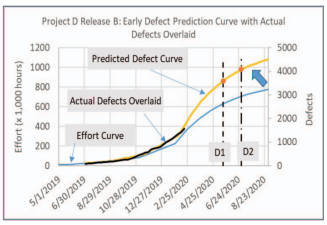

Fig. 9: Early defect prediction with actual defects overlaid

C. Transformation algorithm for early defect prediction

We now have input data (as shown in Figure 8) ready for generating an early defect prediction curve for the current release. We will use the logic (as described in [9]) derived from the Quantile-Quantile (Q-Q) plot, i.e., a defect curve is highly correlated to a leading curve, which is the effort data in this case. The transformation function contains two elements: horizontal shift and vertical shift. In other words, the leading curve (i.e., the effort curve) is transformed by shifting it horizontally and vertically to go through the target defects (TD) at D_1 and D_2 as closely as possible. The transformation function from (x, y) coordinates to (x_{new}, y_{new}) is described in eq. 8 and eq. 9 for horizontal shift and vertical shift respectively.

$$x_{new} = \alpha + \beta x \tag{8}$$

$$y_{new} = \gamma y \tag{9}$$

The parameter α represents the constant delay in weeks from the effort curve to the defect curve, and β represents the additional delay in the defect curve. The parameter γ is determined as a ratio of defects and effort hours used for the best-fitted curve. It means defects per effort hour. It is a non-linear optimization problem with three variables (or parameters) and two data points to minimize the sum of distances between predicted defects and target defects at D_1 and D_2. We can solve this problem iteratively with numerical analysis. It requires three-level loops and several iterations for each level. We need to set up the initial value, the increment, and the maximum number of iterations for each level. Table I shows default values developed by a trial and error method for several different data sets, where DD_2 is the defect density at the delivery date D_2.

Parameter	Initial Value	Increment	Max. Iterations
α	4	-0.5	10
β	0.7	0.05	10
γ	$DD_2 - 0.9$	0.01	10

TABLE I: Default values for the transformation parameters

These numbers can be implemented as separate parameters to be changed according to the required accuracy. Our proposed default values are considered reasonable in practice based on our experience. For each level we keep track of minimum sum of distances (SD) between target defect values (TD) and predicted defect values (PD) at D_1 and D_2. It is calculated as:

$$\text{SD} = w \times |\text{TD}_1 - \text{PD}_1| + (1 - w) \times |\text{TD}_2 - \text{PD}_2| \tag{10}$$

Where w represents a weight factor with a default value of 0.8. Note that target defects (TD) can be computed from 7, and predicted defect values (PD) are calculated using the transformation functions 8 and 9. Note that the transformed curve is no longer aligned with the original x-axis due to 8. We use an interpolation method to calculate exact PD values at D_1 and D_2.

The following three levels of iterations will be performed.

- Step 1: We first begin the iteration at Level 3. We calculate the SD value and compare it with the previous iteration. The SD values at Level 3 (SD_3) are expected to decrease until the minimum value is found. It will stop when the current SD_3 is larger than the previous SD_3. We set the last SD_3 as the $min(SD_3)$ and the values of α, β, and γ.
- Step 2: We set $min(SD_2) = min(SD_3)$ and then move to the next value of Level 2 and repeat the Level 3 operation. The SD values at Level 2 (SD_2) are expected to decrease until the minimum value is found. It will stop when the current SD_2 is larger than the previous SD_2. We set the last SD_2 as the $min(SD_2)$ and the values of α, β, and γ.
- Step 3: We set $min(SD_1) = min(SD_2)$ and then move to the next value of Level 1 and repeat the Level 2 and Level 3 operations. The SD values at Level 1 (SD_1) are expected to decrease until the minimum value is found. It will stop when the current SD_1 is larger than the previous SD_1. We set the last SD_1 as the $min(SD_1)$ and the values of α, β, and γ.

The above iterative procedure will find the global minimum and respective parameter values. We re-calculate the final predicted curve with the final parameter values. Note that the last curve will need to be converted to weekly data via an interpolation method since the transformation shifted the time scale. The final defect arrival curve is shown in Figure 9, where the curve should be closely going through the target defects at D_1 and D_2. We can also overlay actual defect data to validate the early defect prediction, as shown in Figure 9. We have

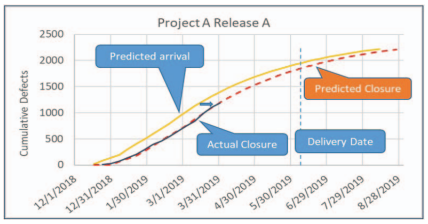

Fig. 10: Closure curve prediction vs. actual closure data

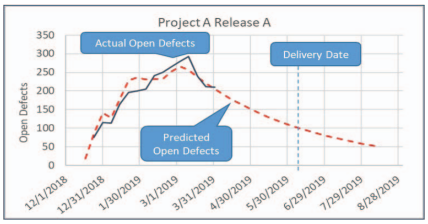

Fig. 11: Open defect curves: Actual defects vs. predicted defects

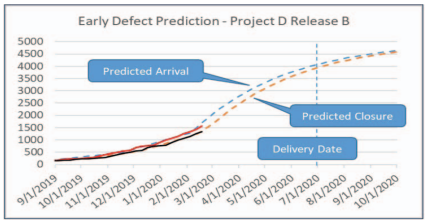

Fig. 12: Actual arrival and closure data with early defect prediction

completed an early prediction for the defect arrival curve. We are now ready to look at a closure curve prediction.

D. Closure curve prediction

Input data are the predicted arrival curve as the leading data and actual closure data, as shown in Figure 10. We will apply the transformation function described in Section IV-C. We have found that the predicted closure curve can be derived by shifting the leading data to the right based on many project data. This is a particular case where $\beta = 1$ and $\gamma = 1$. We set several actual closure data points to find the value α, which provides the best curve. The default value is 2. That is, the last 2 data points are used. In this example, we found $\alpha = 2.3$, which means the closure prediction is derived by shifting the arrival curve by 2.3 weeks. Figure 10 shows the predicted closure curve along with actual data. It visually demonstrates the validity of the algorithm. This shifted value derived from previous release data will be used for early closure prediction.

E. Open curve prediction

One of the most critical metrics in practice is the number of open (or backlog) defects. It represents the defects that are still not closed (or fixed). Ideally, we want all detected defects to be fixed (or corrected) by delivery. The open defect curve can be derived as the difference between the arrival curve and the closure curve, i.e.,

$$\text{Open Defects} = \text{Arrival Defects} - \text{Closure Defects} \quad (11)$$

Using the same data as in Section IV-D, we calculated the open defect curves for actual and predicted values from 11. Figure 11 shows the actual data closely following the predicted curve.

F. Outputs

In Sections IV-C, IV-D and IV-E, we demonstrated the algorithm to generate the following early defect prediction curves without actual data: Predicted arrival curve, Predicted closure curve, and Predicted open curve

These curves are illustrated in Figure 12 for the arrival and closure curves and Figure 13 for the open curve, respectively. They also show actual data for arrival, closure, and open curves overlaid with the predicted curves, respectively. Both actual data are closely following the predicted curves. Using these curves, we can derive key quality metrics.

- % Residual defects: We normalize residual defects by the total defects to use this metric for other projects. It is defined as:

$$\%\ \text{Residual defects} = \frac{\text{Total defects} - \text{Defects found}}{\text{Total defects}} \quad (12)$$

  This metric is used to determine if we find enough defects. Our proposed threshold values based on the experience are as follows: It is acceptable if less than or equal to 15%, at risk if greater than 25%, and warning for in-between.
- % Open defects: We normalize open defects by defects found, which is defined as:

$$\%\ \text{Open defects} = \frac{\text{Total defects} - \text{Defects fixed}}{\text{Defects found}} \quad (13)$$

  This metric is used to determine if we fix enough defects. Our proposed threshold values based on the experience are as follows: It is acceptable if less than or equal to 5%, at risk if greater than 10%, and warning for in-between.

Combining the above two quality metrics, we can now make an intelligent decision about whether the software release is ready for delivery. A sample of metrics data is summarized in Table II.

Will we be ready for delivery on time with acceptable quality ?		
Key Quality Metrics	**D1**	**D2**
% Residual Defects (Will we find enough defects?)	24.30%	13.80%
% Open Defects (Will we fix enough defects?)	13.20%	6.90%

TABLE II: Sample metrics for the release readiness evaluation

V. Early Defect Prediction with Actual Defect Data

As the test activity begins, defect reports start coming in. We want to update the original defect arrival prediction curve with actual data. We will follow the latter part of the process described in Figure 3.

The input data includes: (1) Actual arrival data, and (2) The original defect arrival prediction curve without actual data: This will be the leading data for early defect prediction with actual data.

We use the transformation function with parameters α, β, and γ, as described in Section IV. We introduce another parameter representing the number of most recent weekly data points, n_p, for the transformation. This will help incorporate

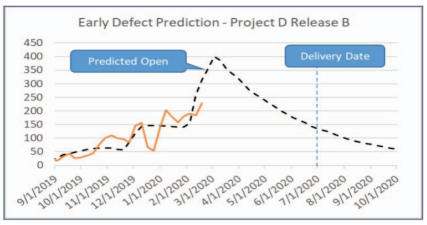

Fig. 13: Actual open defects with early defect prediction

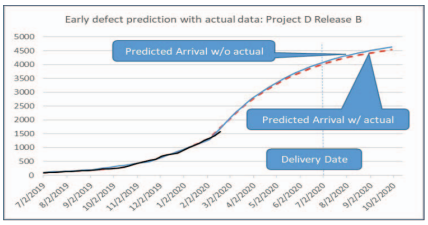

Fig. 14: Early defect prediction with actual data and the leading data

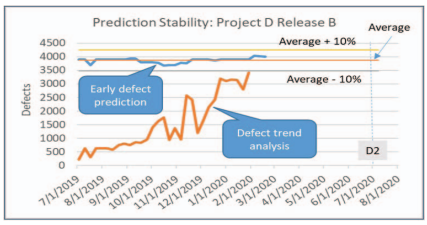

Fig. 15: Prediction stability: early defect prediction vs. defect trend analysis

more recent data. The default value is $n_p = 10$. We will use the last np actual data points for this transformation. Following the same algorithm described in Section 3, the optimum set of α, β, and γ values will be obtained to minimize the sum of distances between actual data and transformed data. The proposed default values are summarized in Table III.

Parameter	Initial Value	Increment	Max. Iterations
α	1	-0.25	4
β	0.9	0.025	4
γ	0.9	0.025	4

TABLE III: Default values for the transformation parameters

The updated arrival predicted curve and actual and leading data are in Figure 14. Note that the predicted curve is very close to the leading data, and both curves are remarkably close to actual data. It implies that the original prediction without actual data is very accurate up to this point in time.

VI. Prediction Stability and Accuracy

We now repeat the procedure described in Section V as new weekly data becomes available. We keep track of the predicted defects at the delivery date, D_2, for each week. A summary of the results is shown in Figure 15. It also shows the average of the predicted defects at D_2 along with +/- 10% limits. It offers a remarkably stable trend over time and an accuracy of less than +/- 10%.

We also want to compare the prediction stability with the defect trend analysis described in Section III. The prediction stability data for the defect trend analysis are overlaid in Figure 15. It is not stable yet. It usually becomes stable several weeks before the delivery date, when the system test begins. Figure 15 demonstrates the power of development and test effort data with previous release data to improve prediction stability and accuracy.

VII. Conclusion

This paper has presented a novel method for an early defect prediction that transforms development and test effort data and learns from previous release data to generate a defect prediction curve. We then extended the defect detection to cover defect closure and open curves. Key quality metrics are addressed for deciding whether the software is ready for delivery. Our proposal is aimed at helping project managers balance development resources and software quality at the delivery date during the planning phase. It improves the prediction stability and accuracy remarkably compared with the defect trend analysis based on multiple curves, which significantly enhances the traditional single curve fitting.

The innovative method has been implemented as the core engine of a new online analytics tool [12], STAR, for software quality assurance. STAR has been built thanks to experiences from engagements with various business groups at Nokia and through learning the pain points of developing and predicting defects in large scale software. We invite interested parties to reach out and try out the tool for their projects.

Acknowledgment

I would like to thank Dr. Rashid Mijumbi, Rory Harpur, Joseph Good, and Michael Okumoto for their significant contributions to the work in this paper.

References

[1] H. Okamura and T. Dohi, Application of EM Algorithm to NHPP-Based Software Reliability Assessment with Generalized Failure Count Data. Mathematics 2021, 9, 985. https://doi.org/10.3390/math9090985

[2] Q. Li, and H. Pham, Modeling software fault-detection and fault-correction processes by considering the dependencies between fault amounts, Applied Sciences (Switzerland), 11(15), 2021 [6998]. https://doi.org/10.3390/app11156998.

[3] A. Nikora, L. Fiondella and T. Wandji, SFRAT – An Extendable Software Reliability Assessment Tool, 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2018, pp. 65-70, doi: 10.1109/ISSREW.2018.00-29.

[4] M. R. Lyu. Handbook of Software Reliability Engineering, Computer Society Press, Los Alamitos, and McGraw-Hill, New York, 1995.

[5] List of software reliability models, Wikipedia, https://en.wikipedia.org/wiki/List_of_software_reliability_models.

[6] CASRE: a computer-aided software reliability estimation tool. 1992. doi:10.1109/CASE.1992.200165.

[7] 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.

[8] K. Okumoto, Experience Report: Practical Software Availability Prediction in Telecommunication Industry, Proceedings of 27th IEEE International Symposium on Software Reliability Engineering (ISSRE), pp. 331-342, 2016, Ottawa, Canada.

[9] R. Mijumbi, K. Okumoto, A. Asthana, J. Meekel, Recent Advances in Software Quality Assurance, 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Oct 2018, pp. 77–82, Memphis, TN, USA.

[10] R. Mijumbi, K. Okumoto, A. Asthana, Software Reliability Assurance in Practice, Chapter in Wiley Encyclopedia of Electrical and Electronics Engineering, 2019.

[11] K. Okumoto, Software Quality Assurance as a Service (STAR): A revolutionary Approach, Two-hour Tutorial at 26th International Conference on Engineering of Complex Computer Systems, Mar 26, 2022, Hiroshima, Japan. http://iceccs2022.xsrv.jp/.

[12] Software Quality Assurance as a Service (STAR), Sakura Software Solutions. https://sakurasoftsolutions.com/.