Digital Engineering-driven Software Quality Assurance-as-a-Service

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

***Abstract*—Recent advances in technology are pushing the limits in computation, networking, storage, and are allowing an increasing number of business processes to be intelligently automated. These advances require, in a large part, the ability to continuously design, develop, test, integrate and deploy high quality software. More than ever, software development teams need ways to plan their release timelines, and to ensure that released software is of the expected quality. While software quality assurance models are well-studied, there remains a significant gap in translating these models to practice. In practice, subject matter experts are required to continuously tweak models and their parameters, and generally perform software reliability analysis using spreadsheets. This usually generates inconsistent and biased results. The results are not easily shared within the software development team, which creates a lack of communication.**

In this paper, we present STAR, a novel cloud-based tool for software quality assurance that is driven by digital engineering principles. The core engine of STAR implements a series of piece-wise exponential models for defect trend analysis. In addition, during early stages of projects where there is not defect data, STAR proposes an algorithm that uses software development effort plans from previous releases to predict defects for a current release. Furthermore, STAR includes prescriptive scenario planning, which allows users to run interactive what-if scenarios and corrective actions needed to meet delivery and quality targets. All these have been implemented in a software-as-a-service model that allows users to create projects and releases, and automatically obtain all the necessary quality metrics to decide whether the release is ready for high-quality delivery. Evaluations of the effectiveness of STAR have been performed using data from large-scale software development projects, as well as through collaborations with large industry and academia partners.

***Index Terms*—Software reliability, digital engineering, software quality, on-time delivery, enterprise and program decision-making, program readiness, prescriptive data analytics, engineering practice, decision analytics, and visualization**

I. Introduction

One of the biggest challenges faced by engineering teams is evaluating software quality and using this analysis's results for readiness-of-release decision-making. Because most software development projects have pre-defined delivery timelines, the challenge involves predicting the number of defects remaining in the software at a given future date and assessing the impact on the overall program of which the software is part. Methods for predicting software defects over time are often called software reliability growth models. These methods typically use defect trend analysis to create a cumulative defect arrival curve and, based on the shape of the curve, apply an appropriate statistical model to fit a curve to the actual data.

The theoretical area of software reliability growth modeling is well studied, with over 220 models and a few tools proposed since the 1970s [1]. Most of these models use a single complex curve to describe an entire defect trend, called a defect trend analysis. Goodness-of-fit is used to compare different models, and none of them address the predictability of the models, which is of utmost importance for practitioners. Applying these models effectively in practice remains generally unsolved. This is mainly because defect data from actual projects are not readily available to researchers in the field. As a result, some assumptions many models make (such as having a developed and stable product) do not hold in practice. Moreover, the models still require specialist knowledge to apply. This is typically performed by subject matter experts using custom spreadsheets which produce inconsistent output and are difficult to share and maintain across teams.

This paper presents STAR[1], a digital engineering tool that transforms software engineering through a data-driven software quality evaluation to support software release decision processes. It is a new cloud-based, real-time tool, STAR, for software quality assurance. The tool is aimed at practitioners who manage software quality and make decisions based on its readiness for delivery. Being web-based and fully automated allows teams to collaborate on software quality analysis across multiple stakeholders, projects, releases, and components.

Figure 1 provides a summary of the main attributes of STAR. It enables transparency for all participants, effective management of teams, precise control of the progress, and holistic monitoring of development results. STAR has been modeled following the Software-as-a-service (SaaS)[2] paradigm, which is a way of delivering applications over the Internet—as a service. Instead of users installing and maintaining software, they access it via the Internet, freeing themselves from complex software and hardware management responsibilities. SaaS has become a standard delivery model for many business applications.

The core innovation of STAR is in its set of statistically sound algorithms that are then used to generate a defect prediction curve for the provided data. We recently developed an innovative data-driven method [3], [4] for generating multiple

[1]https://sakurasoftsolutions.com

Fig. 1: STAR: Software Quality Assurance-as-a-Service

curves to construct a series of piece-wise exponential models. Considering that not all software contents are available at the beginning of the test phase, it makes sense to look at the defect trend for each period during the test phase as a new set of contents becomes available. An exponential model has proven effective when the content is stable, e.g., in the system test. This is achieved through the automated identification of inflection points in the original defect data and their use in generating piece-wise exponential models that make up the final prediction.

STAR implements innovative ways to generate the defect arrival/closure and open curves, expanding the software reliability field beyond the defect arrival prediction. Moreover, during the early days of software development, where no defect data is available, STAR can use the development effort plan to learn from defects of previous software releases to make predictions for the current release. We developed a breakthrough method for transforming the effort data into a defect curve. It enables the generation of a defect arrival curve without actual data during the planning phase. The transformation method is also used to predict the closure curve.

Finally, the tool implements a range of what-if scenarios, enabling practitioners to evaluate several potential actions to correct course. We introduced an interactive method for quantifying corrective actions' quality impact, such as delaying the delivery date, adding more developers to fix defects, adding more testers, and reducing the release content.

STAR will enable companies with in-house or subcontracting software development to achieve high-quality and on-time software delivery. This is the first of its kind in the field. It is available for public use. In particular, STAR can provide answers to these and other related questions to help the software development team deliver quality software on time:

- When will the product be ready to be delivered, and how many defects shall we find?
- How many additional defects could we find from now to delivery?
- Which of our software components are most defective?
- What can we do to improve the software quality? And what if we increased the number of developers or delayed the delivery?
- What if we don't have defect data early on in development? How can we make early quality estimates?

The STAR outputs are presented in state-of-the-art visualization in tables and charts suitable to all users, from developers to executives.

II. Related Work

Foundational SRGMs formulated the problem as a non-homogeneous Poisson process (NHPP) [3], [5], [6]. Two main approaches are typically used: parametric and non-parametric. Non-parametric approaches use software characteristics (such as size, complexity, and development effort), test metrics (such as test cases executed and pass rate), and project milestones as input. In these approaches, the core processing technique is typically machine learning [7].

Parametric approaches take advantage of the dynamic nature of defects during the test as the defect trend continues to change due to the find-fix process. The generic name – software reliability growth modeling (SRGM) – is derived from this process [8]. These approaches rely on mathematical models to capture the defect trend during the test. Due to their data-driven approach, they require a certain amount of data before the prediction becomes stable [9]. Most SRGMs are, in practice, grouped into two: exponential models [9] and S-shaped curve models [10]. Exponential models are simple and rely on easily understood assumptions. They are based on a statistically sound method of parameter estimation. On the other hand, S-shaped models are intuitively understood. Their shape reflects the integrated effects of software structure and learning factors on the testing process. A major difference between exponential models and S-shaped curve models is in the early test phase. Exponential models ignore defects from the early test phase, while S-shaped models attempt to capture these defects. Our experience shows that capturing early defects, even with S-shaped models, is difficult.

However, the complex nature of such curves presents challenges in attempts to establish statistical methods for estimating their parameters. As a result, the curves are usually generated manually by subject matter experts. Moreover, most S-shaped models cannot explain an early part of defect data if focused on the latter part of the defect data. Generally, a single curve approach does not appear sufficient to describe the entire defect data. Even though this has been a well-studied subject over the last 50 years [11], it continues to be challenging in practice. One possible solution to this is to use multiple exponential curves for the different phases of the development process, which is the method implemented in STAR.

BRACE [12], [3], [13], [8], [14] is a cloud-based software reliability assurance tool that was developed and used by software development groups at Nokia. The tool includes models for defect prediction at various stages of the development lifecycle. The code for the tool has been open-sourced and has been the main motivation of the innovative work in STAR. This paper will significantly enhance the functionality in STAR from an automation, computational efficiency, user-friendliness, and cybersecurity perspective.

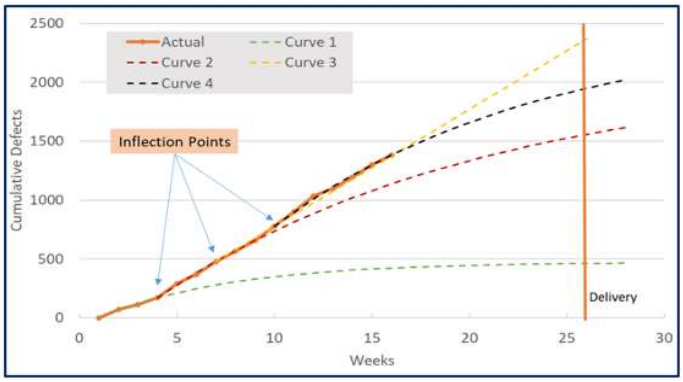

Fig. 2: A series of the piece-wise exponential model

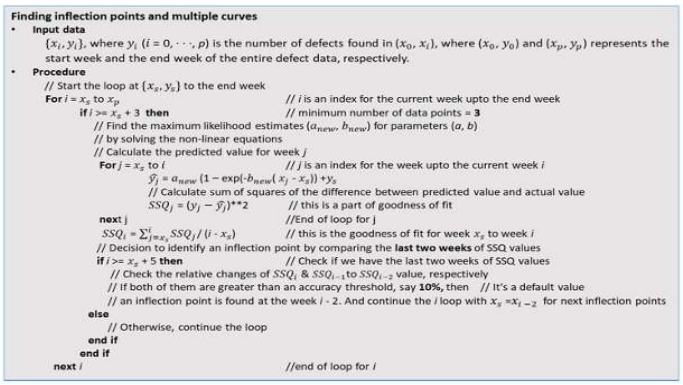

Fig. 3: Algorithm for finding inflection points and multiple curves

III. STAR Overview

A. Defect Trend Analysis

This section addresses our new innovative method for defect trend analysis. The earlier version of the technique was introduced in [8], [13]. Since then, we have significantly improved the algorithm. There are two parts: multiple curve generation and adjustment for the last curve.

The first step is finding inflection points or points 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 based on 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 one 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 2.

The algorithm in Fig. 3 provides a high-level procedure for finding inflection points and multiple curves. It is a series of piece-wise exponential models with a general form of equation 1

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

Note $m_i(t_i)$ is the cumulative defect data at time or week t_i for period i. Parameters a_i and b_i represent total defects and detection rate for the period i, respectively. 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 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.

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' effectiveness illustrated in Figure 2. 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. Refer to [4] for detailed adjustment.

B. Early Defect Prediction

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 to stabilize the prediction. It is not appropriate for defect prediction during the planning phase when actual defect data is unavailable, called 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 that effort data is best suited for this purpose. 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.

1) Transformation Algorithm: We have developed an innovative method for transforming the effort curve into the defect curve, which goes through the target values at D1 (delivery for trial) and D2 (delivery for deployment). See [4] for technical details. 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 at D1 and D2 as closely as possible. In other words,

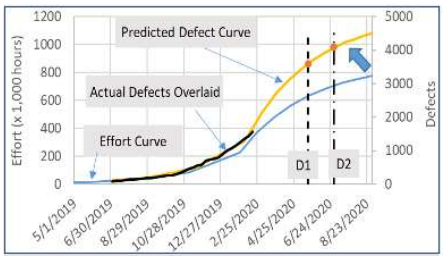

Fig. 4: Early defect prediction with actual defect data

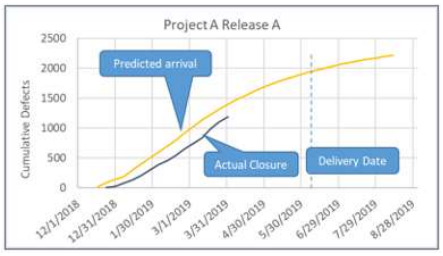

Fig. 5: Closure curve prediction vs. actual closure data

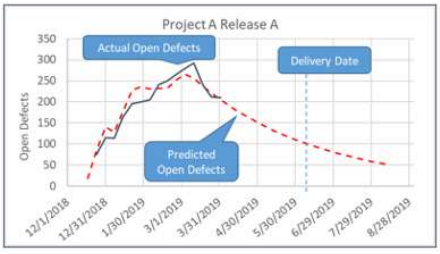

Fig. 6: Open defect curves: Actual vs. predicted

the transformation function from (x, y) coordinates to (x_{new}, y_{new}) is described in equations 2 and 3 for horizontal and vertical shifts, respectively.

$$x_{new} = \alpha + \beta x \tag{2}$$

$$y_{new} = \gamma y \tag{3}$$

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 a ratio of defects and effort hours used for the best-fitted curve. The unit is defects per effort hour. The problem is formulated as 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 D1 and D2. We can solve this problem iteratively with numerical analysis. It requires three-level loops and several iterations for each level. A detailed algorithm is described in [4]. Once actual data becomes available, we will adjust the algorithm to incorporate actual data (in addition to the target defects at D1 and D2). The adjustment is based on the number of actual data points, say the last four weeks' data as a default. This will maintain the original early prediction during the early phase of testing. And actual data will become a dominating factor in the defect prediction as we approach the delivery date. Figure 4 illustrates the effort curve transformed into the predicted curve, which closely goes through the target values at D1 and D2. It also shows the early defect prediction and overlays actual defect data, which is a near-perfect fit.

C. Closure Curve Prediction

Input data are the predicted arrival curve as the leading and actual closure data, as shown in Figure 5. We will apply the transformation function described in Section 4.2. 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 5 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.

D. Open Curve Prediction

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

$$\text{Open Defects} = \text{Arrival Defects} - \text{Closure Defects} \tag{4}$$

Figure 6 shows the actual open defect data closely following the predicted curve.

E. Key Quality Metrics

We demonstrated the algorithms to generate arrival curves with two cases: 1) defect trend analyses described in Section 4.1 and early defect prediction curves with and without actual data described in Section 4.2. In addition, we presented closure and open prediction curves in Sections 4.3 and 4.4, respectively. Both actual data closely follow the arrival, closure, and open predicted curves. Using these curves, we can derive key quality metrics which will be used for the evaluation of delivery readiness based on the quality targets.

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

$$\%\ \text{Residual defects} = \frac{\text{Total Defects} - \text{Defects found}}{\text{Total defects}} \tag{5}$$

This metric is used to determine if we will find enough defects. Our proposed threshold values based on the experience are as follows: It is acceptable (green) if less than or equal to 15%, at risk (red) if greater than 25%, and warning (yellow) for in-between.

2) % Open Defects: We normalize open defects by defects found. In other words,

$$\%\ \text{Open defects} = \frac{\text{Defects found} - \text{Defects fixed}}{\text{Defects found}} \tag{6}$$

This metric is used to determine if we will fix enough defects. Our proposed threshold values based on the experience are as follows: It is acceptable (green) if less than or equal to 5%, at risk (red) if greater than 10%, and warning (yellow) for in-between. Combining the above two quality metrics, we can now decide whether the software release is ready for delivery. A sample of metrics data is summarized in Table 7 Note that the Color coding of green, yellow, and red implies "Acceptable," "Warning," and "At risk."

Will we be ready for delivery on time with acceptable quality?

	D1	D2
Percentage Residual Defects *(Will we find enough defects?)*	25.8%	12.1%
Percentage Open Defects *(Will we fix enough defects?)*	10.6%	4.2%

Fig. 7: A sample of key quality metrics for the release readiness evaluation

IV. Implementation

A. Overview

User input data are defect data, development, test effort data, and project milestone dates such as the test start date, delivery date for customer trial (D1), and delivery date for deployment (D2). The input data can be entered manually via CSV files or imported directly from the customer database. The core engine will take over the computation and prediction, including preprocessing user input data suitable for central processing. It implements innovative algorithms, which go through the concept, prototype, proof of concept, trials, and productization. Technical details are discussed in [4]. Outputs are presented via our state-of-the-art user-friendly interface with visualization with many info buttons explaining the definitions of the terms and color-coding.

B. STAR System Architecture

STAR automates the entire process of input data extraction, pre-processing, core analytics, and post-processing. Figure 8 presents STAR's high-level architecture based on the AWS platform. STAR collects data from multiple defect logging tools using application programming interfaces (such as Flask) and stores the data required in a unified format in two databases: PostgreSQL & NoSQL (DynamoDB), improving performance. This data is then pre-processed before being stored in databases. Specifically, pre-processing involves aggregating the data into a weekly or specified time frame and preparing necessary input data for the core engine, which performs prediction. An essential pre-processing function is to keep the data consistent from project to project, and within a project, from release to release. As an example, the pre-processing may include:

- Unifying database field and value mappings since different projects/databases use field names for the same properties, e.g., priority vs. severity.
- Some field values may need to be mapped to specific values, e.g., a defect is assigned to a geographic location vs. an organizational unit.
- Checking for defect properties to identify duplicates or those from other releases to facilitate quality vis-à-vis project management.

The core engine is developed using Python Scientific Stack, and the user interface is based on Javascript ES6 (React). Terraform AWS is used as the infrastructure as code (IaC).

C. Demo Data

Actual defect and effort datasets have been generated based on over 50 years of experience working with real projects.

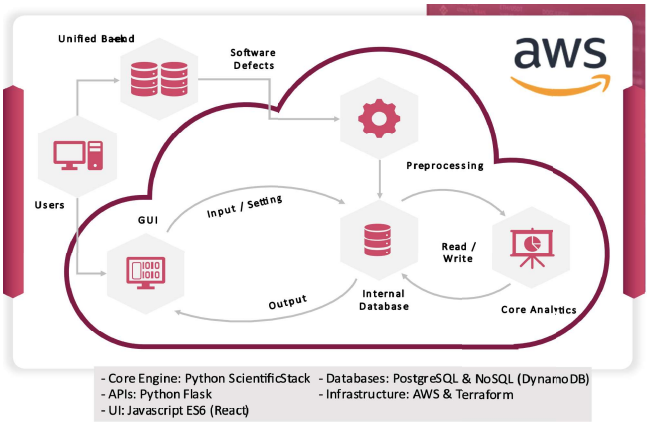

Fig. 8: STAR System Architecture

Based on proprietary analytical models, arrival, closure, and open defect data predictions will be produced [4]. State-of-the-art visualization will help you understand STAR results easily. A simple, user-friendly user interface will make it easy to use STAR. There are four sets of project data, called Projects A – D, and each project has a few releases. They encompass a variety of project sizes from small to large and various shapes of defect trends. We will use them to demonstrate the power of automated defect prediction for all cases, i.e., end-users need no parameter adjustments.

- Project A: This data set presents a typical scenario where we want to know whether the current release is ready for delivery several weeks before delivery.
- Project B: This data set presents several snapshot views for the same release. It demonstrates the importance of continuous defect prediction as the defect trend changes depending on test progress.
- Project C: This data set contains three releases within the same project to demonstrate release over release.
- Project D - This data set demonstrates an early defect prediction for Release B using defect and effort data from the previous release, Release A.

D. Preprocessing

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

- 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. The normalized development effort curve represents how the software content is tested over time.

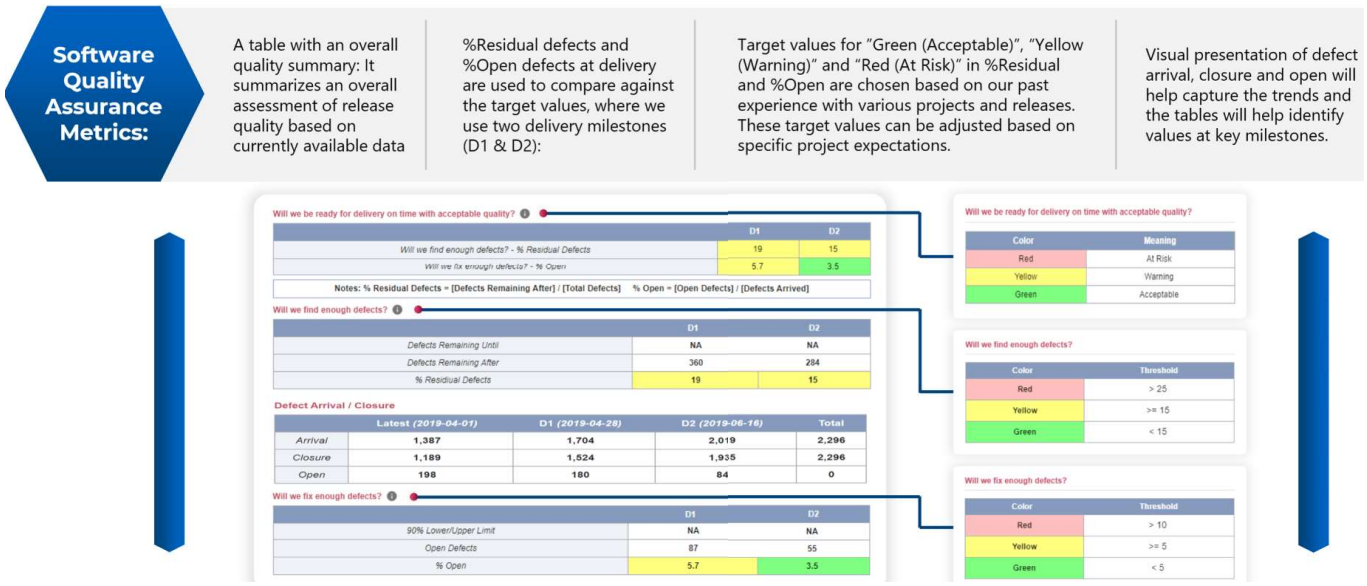

Fig. 9: Explanations of Executive Summary View

- Adjusting the tail end of the normalized development effort curve: We must adjust the tail end since it shows a rapidly flattening trend. This is because test effort represents internal test activities. The adjustment will compensate for customer defects. Defect data usually includes those found by customer testing and operation. The adjustment can be made using the trend analysis algorithm discussed in Section 4.1. The final adjusted effort curve is shown in blue in Fig. 4.
- Calculation of defect density: We are now ready to calculate defect densities at D1 and D2 using the defect curve and the adjusted normalized effort curve. An interpolation method identifies values at D1 and D2, respectively. We can then calculate defect density as defects over effort hours.
- Target defect values at the delivery date: We perform the steps described above for the effort data from the current release to derive the adjusted normal development effort curve. Using the defect densities, we can compute target defects at D1 and D2 for the current release as effort hours multiplied by defect density. The target defects are shown in Fig. 4.

V. Sample Visualisations

A. Defect Trend

STAR has a defect trend menu in the "Executive Summary" View represents a defect prediction based on actual defect trend data. STAR automatically identifies the inflection points using the above mentioned algorithm and generates a series of piece-wise exponential models. Although the algorithm is designed for software defects at the release level, it is flexible for accommodating individual components, severities, customer-based defects, hardware failure data, cybersecurity issues, and many other applications. The defect filter box in

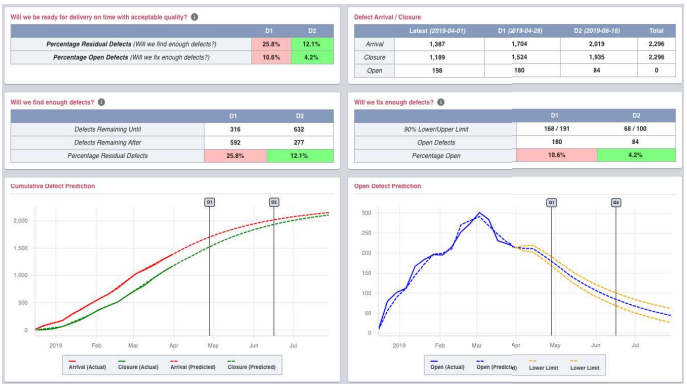

Fig. 10: Executive Summary View

the Executive Summary View demonstrates the power of the algorithm for components and severities, where each defect data set varies in size and shape as small as less than 100 defects. All cases show a perfect fit.

B. Executive Summary

Figure 10 shows critical quality metrics with tables, charts, and self-explanatory info buttons. As Table 9 highlights, STAR provides a few info buttons to explain the interpretations.

- % Residual defects to answer the question: Will we find enough defects?
- %Open defects to answer the question: Will we fix enough defects?
- Defects at Delivery for trial (D1) and delivery for deployment (D2)
- Arrival/closure and open defect charts help capture the trends and identify the values at critical milestones
- Defect filter(s) allows a selection of components and severity to demonstrate the power of automatic prediction

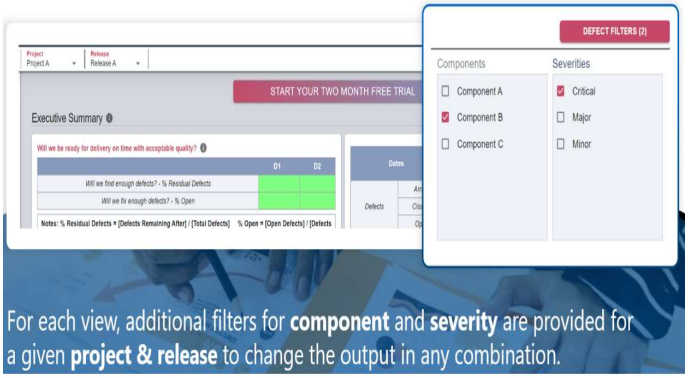

Fig. 11: Defect prediction by component and severity

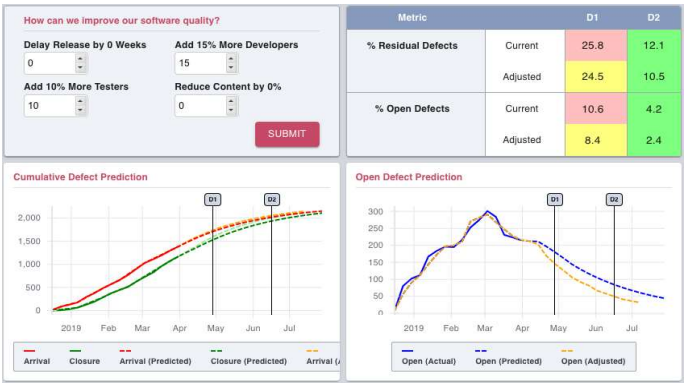

Fig. 12: STAR interactively quantifies the quality improvements for corrective actions

in various ways. Figure 11 highlights the defect filters. For each view, additional filters for component and severity are provided for a given project & release to suit all users, from developers to executives.

C. Prescriptive Analytics

Real-time view of quality impact by a selected action provides two corrective actions: Delay delivery date and Additional developers for bugfixes. It will interactively quantify the improvement in %Residual and % Open for respective options, as highlighted in Fig. 12

- Arrival/closure: Weekly view of defect arrival and closure curves
- Defects by component: Defects trend breakdown by component help identify a problematic component.
- Defects by severity: Defect trend breakdown by severity to help focus on critical and major defects. See Fig. 13 for a sample output.
- Release over release: Defect trend view of the last three releases, normalized by the delivery date
- Prediction stability: Weekly changes in prediction at the delivery date over time for evaluating the stability and accuracy of prediction – demonstrates the accuracy of STAR prediction very close to actual data many weeks before the delivery date
- Early prediction (Leading data (PPM)) shows an early defect prediction arrival curve generated without actual defect data with +/- 10% limits and target defects at D1 and D2, as shown in Figure 14. The target values are derived using defect and effort data from the previous release. This is available only for Project D Release B.

D. Early Defect Prediction

STAR implements the innovative algorithm described above to automatically generate the defect prediction curve without actual defect data. It transforms the effort data (as leading data) into a defect curve. The outputs of early defect prediction are shown in the following two STAR menu items:

- Leading data (PPM): This menu represents an early defect prediction without actual defect data during the planning phase. It is based on previous release data (Release A) with software development and test effort data. This is available only for Release B.
- Leading Data (IPM): This menu represents a defect prediction with actual defect data from Release B using Leading data (PPM) output as the leading data.

Fig. 13: Defect breakdown by component

Figure 14 shows the early defect prediction curve derived in III-B with actual defect data overlaid. Figure 15 provides arrival/closure and open curves with actual defect data. Together, it enables a visual inspection of the prediction accuracy. Figure 16 shows the stability of early prediction with actual defect data. We also overlaid the stability of defect trend analysis on top of the chart. You will see a significant difference. It is the power of the early defect prediction with effort data in improving the prediction stability and accuracy.

VI. Conclusion

We introduced a revolutionary online tool for software quality assurance. STAR answers frequently asked questions and more to help project managers make intelligent decisions about development resources and release quality at delivery. STAR provides a simple, user-friendly interface with state-of-the-art visualization. The core engine implements a few innovative statistically sound analytics, thoroughly tested using actual project data. It provides stable and accurate defect prediction by incorporating the previous release's development and test effort data. A series of piece-wise exponential models achieve

Fig. 14: Early arrival defect prediction with target values at D1 and D2, actual defect data overlaid

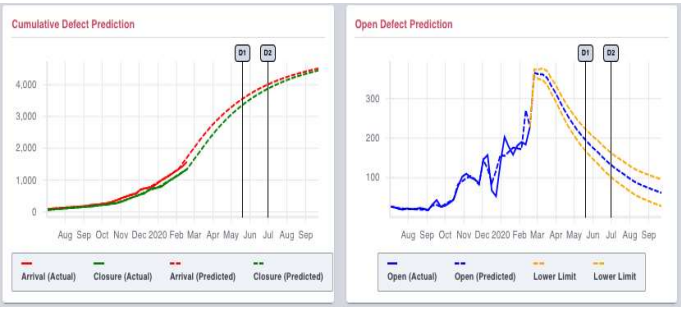

Fig. 15: Early prediction for arrived/closed/open defect curves with actual data overlaid

prediction stability and accuracy several weeks before delivery. It also provides an excellent goodness-of-fit compared to traditional single-curve fitting models. Since it is a cloud-based tool, it is available anytime, anywhere, and to anybody. STAR is at the forefront of a digital engineering tool for software reliability engineering. It is available for public use. Since the algorithm is made flexible, STAR can be used to predict hardware failures. It enables an analysis of both software and hardware reliabilities at the same time.

Acknowledgment

The author would like to thank Dr. Rashid Mijumbi, Rory Harpur, Joseph Good, Mike Okumoto and Andrew Raposo for their contributions to STAR, and to this paper.

References

[1] John D Musa. A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, SE-1(3):312–327, 1975.

[2] Ma Dan. The business model of "Software-As-A-Service". In *Proceedings - 2007 IEEE International Conference on Services Computing, SCC 2007*, 2007.

[3] Kazuhira Okumoto, Abhaya Asthana, and Rashid Mijumbi. BRACE: Cloud-based software reliability assurance. In *Proceedings - 2017 IEEE 28th International Symposium on Software Reliability Engineering Workshops, ISSREW 2017*, 2017.

[4] Kazuhira Okumoto. Early Software Defect Prediction: Right-Shifting Software Effort Data into a Defect Curve. In *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 43–48, 2022.

[5] Hoang Pham and Xuemei Zhang. NHPP software reliability and cost models with testing coverage. *European Journal of Operational Research*, 145(2), 2003.

Fig. 16: Stability and accuracy of early defect prediction vs. defect trend analysis

[6] Xuemei Zhang and Hoang Pham. Software field failure rate prediction before software deployment. *Journal of Systems and Software*, 79(3), 2006.

[7] David D. Hanagal and Nileema N Bhalerao. Literature Survey in Software Reliability Growth Models. In *Software Reliability Growth Models*, pages 13–26. Springer Singapore, Singapore, 2021.

[8] Rashid Mijumbi, Kazuhira Okumoto, Abhaya Asthana, and Jacques Meekel. Recent Advances in Software Reliability Assurance. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 77–82, 2018.

[9] P K Kapur, Shakti Bhushan, and Said Younes. An exponential SRGM with a bound on the number of failures. *Microelectronics Reliability*, 33(9):1245–1249, 1993.

[10] Shigeru Yamada, Mitsuru Ohba, and Shunji Osaki. S-Shaped Reliability Growth Modeling for Software Error Detection. *IEEE Transactions on Reliability*, R-32(5):475–484, 1983.

[11] Amrit L. Goel and Kazu Okumoto. Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *IEEE Transactions on Reliability*, R-28(3), 1979.

[12] Rashid Mijumbi, Kazu Okumoto, and Abhaya Asthana. Software Reliability Assurance in Practice. In *Wiley Encyclopedia of Electrical and Electronics Engineering*. 2019.

[13] Kazu Okumoto, Rashid Mijumbi, and Abhaya Asthana. Software Quality Assurance. In Mohammad Abdul Matin, editor, *Telecommunication Networks*, chapter 3. IntechOpen, Rijeka, 2018.

[14] Rashid Mijumbi, Kazuhira Okumoto, and Abhaya Asthana. Towards automated, end-to-end software defect prediction. In *Proceedings - 25th ISSAT International Conference on Reliability and Quality in Design*, 2019.