STAR:An Innovative Data-Driven Solution for Software Quality Assurance

Kazuhira Okumoto

Abstract Imagine a world where software quality assurance is effortlessly managed using a single digital tool, eliminating the need for complex models and technical expertise. Despite numerous software reliability models developed over the years, practical applications remain challenging. Enter STAR, a revolutionary digital engineering tool offering zero-touch automation, advanced analytics, and intuitive visualizations on a cloud-based SaaS platform. STAR's innovative approach utilizes multiple curves, overcoming challenges posed by traditional models. This chapter delves into STAR's unique features, system architecture, and practical applications through real-world examples, showcasing its effectiveness in software quality assurance.

1 Introduction

1.1 Digital transformation

Software-as a service (or SaaS) is a way of delivering applications over the Internet as a service. Instead of installing and maintaining software, you simply access it via the Internet, freeing yourself from complex software management. SaaS has become a common delivery model for many business applications [1-2]. Picture a world where professionals no longer maintain numerous cumbersome spreadsheets for each release and project. Imagine a scenario where data from defect logging systems is effortlessly gathered, cleaned, and transformed into insightful analytics on software reliability. Envision an environment where quality assurance managers and decision-makers continuously monitor a broad array of software reliability metrics to make well-informed launch decisions. Envisage a state where software reliability

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

Probability and Statistical Models in Operations Research, Computer and Management Sciences, edited by S. Nakamura, et al, Springer, 2024, pp 195 - 220

is analyzed using a single digital tool, eliminating the need for technical experts to create complex models. In this world, anyone can create hypothetical scenarios to aid in launch decision-making. The successful commercialization of this technology promises a world where launch decisions rely on precision and foresight. The software reliability market landscape is on the brink of transformative change, offering a compelling opportunity for innovation.

1.2 Software development process with defect find-fix process

The software release incorporates new features or functionalities developed through a structured development process that includes requirement specifications, software design, coding, and testing. Fig 1 illustrates the different stages of the software lifecycle. After the testing phase concludes, the software undergoes acceptance testing at a customer site before being deployed for commercial use. Internal testers and customers identify software defects, also known as faults or bugs, during this process. The software's quality improves as defects are discovered and resolved throughout internal and customer tests and operations. Due to the overlapping nature of these steps, the development team typically addresses fixes for defects found in internal tests and customer evaluations after the initial delivery to the customer site. However, defects may still emerge during the operational phase after commercial deployment, some of which can lead to system outages or failures in the field.

1.3 Challenges in software reliability modeling

The challenges faced by engineering teams in evaluating software quality and making release decisions are multifaceted, primarily due to the constraints imposed by predefined delivery deadlines. One of the pivotal tasks in this context is predicting the remaining defects in software at a future date and comprehending how these defects might impact the overall program. To address this, various methods have been developed over the years, with software reliability growth models (SRGMs) being at the forefront. These models work by analyzing defect trends, essentially creating a cumulative defect arrival curve.

Despite the extensive development of more than 220 SRGMs [3-16] since the 1970s, their effective application in practical scenarios remains an unresolved challenge. One of the significant hurdles is the limited access to real project data, making it difficult to validate these models against actual software development situations. Additionally, there is a heavy reliance on specialist knowledge, which further hampers the practical implementation of these models.

Currently, experts in the field resort to using custom spreadsheets for their analyses. While this approach allows for flexibility, it gives rise to inconsistent outputs and creates challenges in collaboration and maintenance across different teams within an

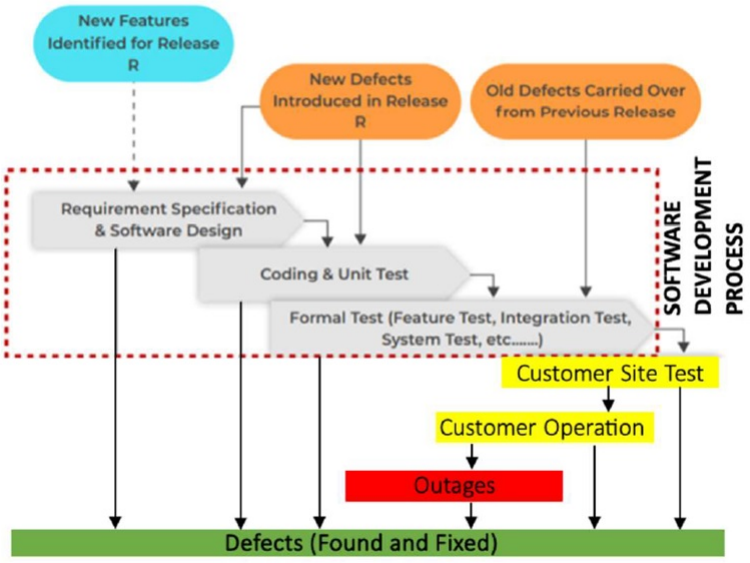

Fig. 1: A sample software lifwe cycle with defect injection and find-fix processes

organization. Moreover, existing models predominantly focus on defects identified during the testing phase, neglecting the broader spectrum of issues that might arise in different stages of software development.

A common approach employed by these models is using single complex curves, such as exponential and S-shaped models, for example, Weibull, gamma, and logistic, to describe the entire defect trend. This method, known as defect trend analysis, simplifies the intricate nature of software defects into overarching patterns. However, the challenge lies in determining the predictability of these models. While goodness-of-fit metrics are utilized to compare different models, they often fall short in addressing the models' ability to predict future defects accurately.

In essence, the current state of evaluating software quality and making release decisions is marred by a lack of comprehensive, practical, and predictive models. The challenges emanate from limited access to real project data, the reliance on specialized knowledge, and the use of oversimplified models. Addressing these issues is imperative for engineering teams to enhance the accuracy of software quality evaluations, meet delivery deadlines, and ultimately ensure the success of their software development endeavors.

To enhance their practical applicability, SRGMs should be capable of addressing common inquiries in the realm of software quality assurance.

- Use Case #1: Will we find enough defects?
- Use Case #2: Will we fix enough defects?

- Use Case #3: Will we be ready for delivery on time with acceptable quality?
- Use Case #4: What if we don't have defect data early in development? How can we make early quality estimates?
- Use Case #5: What can we do to improve the software quality? And what if we increased the number of developers or delayed the delivery?
- Use Case #6: When can we start using the prediction? Are predicted values stable and accurate?
- Use Case #7: Which of our software components exhibits the highest defect rate?

1.4 A revolutionary approach to software quality assurance

Recently, a novel data-driven method was devised [17 - 20] to create multiple curves, enabling the development of a series of piece-wise exponential models. Considering that not all software components are accessible at the commencement of the testing phase, it is logical to examine the defect trends for each period within this phase as new components become available. An exponential model has been proven effective during phases of content stability, such as system testing.

In this chapter, we present a groundbreaking tool for software quality assurance known as STAR. Sect.2 will delve into specific aspects of STAR, providing an overview, detailing its system architecture, and outlining the input data utilized. The innovative predictive analytics integrated into the core engine of STAR will be elucidated in Sect.3. Furthermore, Sect.4 will showcase STAR's efficacy through various use cases based on the frequently asked questions mentioned in Sect.1.3, illustrating its practical application in software quality assurance. Through these examples, you will gain insight into how STAR surmounts existing challenges in this domain.

2 Understanding STAR

2.1 STAR Overview

STAR stands out as a digital engineering tool, as illustrated in Fig 2. It provides a cloud-based service featuring zero-touch automation, cutting-edge analytics, user-friendly visualizations, and interactive what-if scenarios on a Software as a Service (SaaS) platform. Specifically designed for reliability practitioners and bottleneck detectives, STAR revolutionizes data-driven software reliability analysis. The tool's exceptional data-driven analytics have been validated through empirical demonstrations using diverse datasets, underscoring its effectiveness.

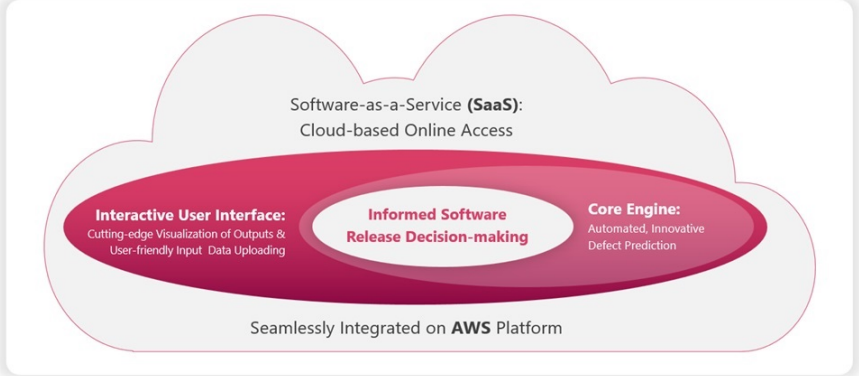

Fig. 2: STAR:A digital engineering tool for software quality assurance.

By adopting the SaaS model, STAR offers users seamless online access, eliminating the need for cumbersome software downloads and ensuring a streamlined user experience. We aim to democratize advanced analytics; automation reduces dependence on domain experts, making invaluable insights accessible to a broader audience. Intelligent visualization capabilities effortlessly transform complex data outputs into understandable formats, expediting decision-making processes.

The earlier version [17-18] of the STAR core engine has gone through prototype, proof of concept, trial, and productization. The algorithm has been redesigned and enhanced with additional features. A new tool [19-20] with an improved core engine was developed on an AWS platform with a simplified user interface and visualization. It's made available for public use.

2.2 Architecture

STAR streamlines the entire process, encompassing input data extraction, pre-processing, core analytics, and post-processing. Fig 3 illustrates the high-level architecture of STAR, built on the AWS platform. To gather data from various defect logging tools, STAR utilizes application programming interfaces (such as Flask) and stores this data in a unified format in two databases: PostgreSQL and NoSQL (DynamoDB), enhancing overall performance. Before being stored, this data undergoes pre-processing. Pre-processing entails aggregating the data into weekly or specified time frames and preparing the necessary input for the core engine, which conducts predictions.

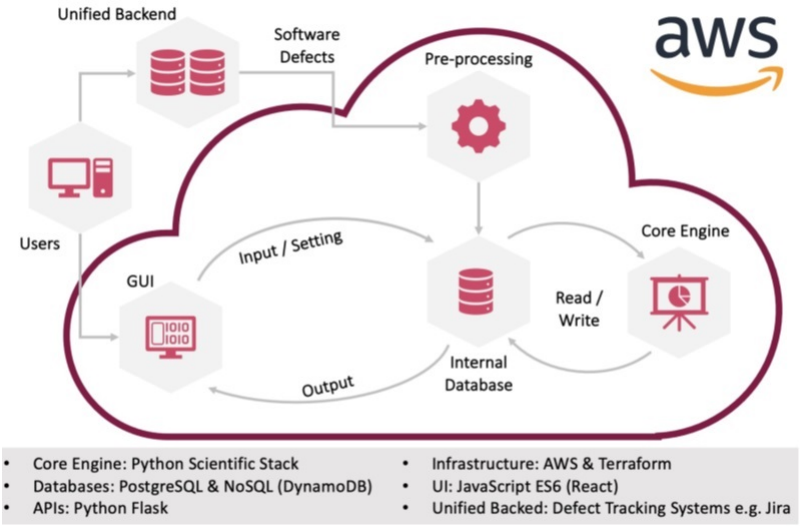

Fig. 3: STAR system architecture.

One crucial aspect of pre-processing is ensuring data consistency across projects and within a project, from one release to another. For instance, pre-processing may involve:

- Standardizing database field and value mappings, as different projects or databases might use different field names for the same properties (e.g., priority vs. severity).
- Mapping certain field values to specific values, such as assigning a defect to a geographic location instead of an organizational unit.
- Checking defect properties to identify duplicates or those from other releases, aiding in quality assessment and project management.

The core engine is developed using the Python Scientific Stack, while the user interface is built on JavaScript ES6 (React). Terraform AWS serves as the infrastructure as code (IaC) for the system.

2.3 Input data

STAR provides an intuitive menu-driven interface. To navigate to the release configuration page, click on ‘User Inputs’ and then select ‘Release Config,’ as shown in Fig4. This page is divided into two sections: release parameters and defect data uploading. In the release parameters section, you are required to input milestone dates (start date, D1 (delivery for trial), and D2 (delivery for deployment) dates).

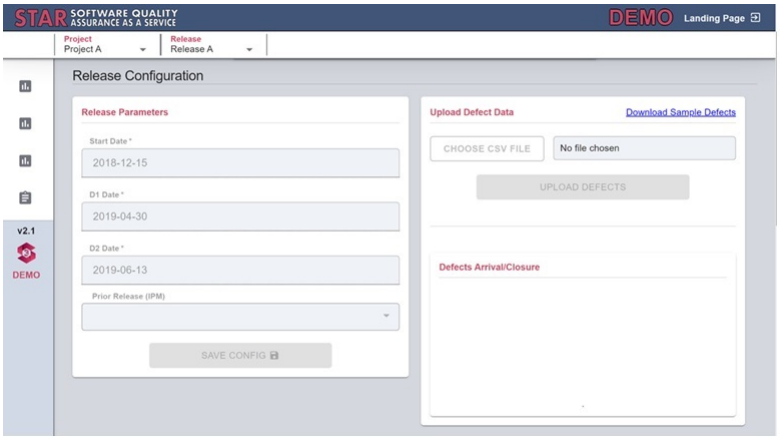

Fig. 4: STAR input data page.

defect_id	component	severity	arrival_date	closure_date
B1-1	Component A	Major	11/20/2018	
B1-2	Component G	Major	11/30/2018	
B1-3	Component D	Major	12/5/2018	12/11/2018
B1-4	Component G	Critical	12/13/2018	
B1-5	Component B	Major	12/15/2018	12/27/2018
B1-6	Component C	Major	12/15/2018	
B1-7	Component A	Major	12/15/2018	
B1-8	Component G	Critical	12/16/2018	
B1-9	Component A	Major	12/16/2018	12/20/2018
B1-10	Component B	Major	12/17/2018	12/21/2018
B1-11	Component B	Major	12/17/2018	12/27/2018
B1-12	Component E	Major	12/17/2018	
B1-13	Component B	Major	12/17/2018	12/25/2018
B1-14	Component B	Major	12/17/2018	12/25/2018
B1-15	Component B	Major	12/17/2018	
B1-16	Component C	Critical	12/17/2018	12/19/2018
B1-17	Component B	Major	12/17/2018	
B1-18	Component B	Critical	12/17/2018	
B1-19	Component B	Major	12/18/2018	
B1-20	Component B	Major	12/18/2018	12/24/2018
B1-21	Component B	Critical	12/18/2018	

Fig. 5: STAR template for defect data.

For defect data uploading, click on the ‘Download Sample Defects’ link to access a template, demonstrated in Fig 5. After creating your data file in CSV format and following the provided instructions, use the ‘CHOOSE CSV FILE’ box to select and upload your file. By clicking the ‘UPLOAD DEFECTS’ box, the upload process will commence, and STAR will display your actual data below the box for your review. The same process applies to effort data.

2.4 Demo data

The defect and effort datasets have been created leveraging more than 50 years of experience gained from real-world projects. Predictions for defect arrival, closure, and open defect data will be generated based on proprietary analytical models. The use of cutting-edge visualization tools ensures that interpreting STAR results is effortless. STAR offers a straightforward and user-friendly interface, simplifying the overall user experience.

The dataset comprises four distinct sets of project data, labeled as Projects A to D, each containing multiple releases. These projects vary widely in size, ranging from small to large, and exhibit diverse patterns in defect trends. These datasets will be used to illustrate the effectiveness of automated defect prediction across all scenarios. Importantly, users will not need to make any parameter adjustments, emphasizing the simplicity of the process.

3 Core engine

In this section, we will outline three fundamental techniques employed by the STAR core engine for predicting defect arrival and closure, showcasing its innovative analytics in the realm of defect prediction.

3.1 Piece-wise exponential models

3.1.1 An exponential NHPP model and the maximum likelihood estimates

Several methods, as detailed in Sect.1, have been investigated, predominantly based on single-curve models. The majority of SRGM models hinge on the Non-Homogeneous Poisson Process (NHPP) concept [15]. To comprehend an NHPP model, envision a limited number of defects, say a. Each defect is identified at time, t, following a cumulative distribution function, F(t). For the defect detection process, N(t), the likelihood of discovering n defects by time t is commonly expressed as a binomial distribution, as depicted in equation (1).

$$P\{N(t) = n\} = \binom{a}{n} F(t)^n \{1 - F(t)\}^{a-n} \tag{1}$$

In practice, the value of a is large, and therefore, we can approximate (1) by a Poisson distribution with the mean value function, m(t), as given in (2).

$$P\{N(t) = n\} = m(t)^n \exp\{-m(t)\}/n! \tag{2}$$

This NHPP is a generalized version, indicating that incremental defect data, like weekly defects, conform to a Poisson distribution with a defined mean value function aligned with the interval. Subsequently, we can create the likelihood function for statistical parameter estimation. It's important to recognize that $m(t) = aF(t)$ signifies the average count of defects detected by time t.

To illustrate, an exponential model can be characterized as an NHPP with the following mean value function:

$$m(t) = a\{1 - \exp(-bt)\} \tag{3}$$

If $b > 0$, $m(t)$ stabilizes exponentially, converging towards $a > 0$. When $b \to 0$ and $a \to \infty$, $m(t)$ becomes a linear function, representing a stationary Poisson process. Conversely, if $b < 0$ and $a < 0$, $m(t)$ values increase exponentially. While b is mostly greater than 0, there are instances where $b \to 0$ during site test and in-service periods and $b < 0$ in early test phases. It's worth noting that the fundamental assumption of a finite number of defects is violated when $b = 0$ or $b < 0$. However, these cases prove useful in explaining diverse trends during individual test periods within the same release.

The NHPP assumption is employed to apply the statistical method of maximum likelihood for estimating model parameters a and b, as described below.

The maximum likelihood method is a widely used statistical approach for estimating the parameters a and b for a given set of defect data. Typically, the data set is represented as(x_i, y_i), where $y_i (i = 0, \ldots, p)$ denotes the number of defects found in the interval (x_0, x_i). Maximum likelihood estimates for a and b are selected to maximize the likelihood function and obtained by solving nonlinear equations as outlined below. It is important to note that we use the term 'estimates' when parameter values are derived from actual data, while the term 'prediction' is used to address future values based on current data and other sources. The likelihood function for a and b is derived from (2) as follows:

$$L(a, b; y_1, \ldots, y_p) = \prod_{i=1}^{p} m(x_i - x_{i-1})^{y_i - y_{i-1}} \exp\{-m(x_i - x_{i-1})\}/(y_i - y_{i-1})! \tag{4}$$

Following some algebraic manipulation involving the partial derivatives of the log-likelihood function (4) with respect to a and b and setting to zeros, we obtain the subsequent pair of equations:

$$a = \frac{y_p}{1 - \exp(-bx_p)} \tag{5}$$

$$\sum_{i=1}^{p} \frac{A_i}{B_i} - C = 0 \tag{6}$$

where the values of A_i, B_i, and C are determined as follows:

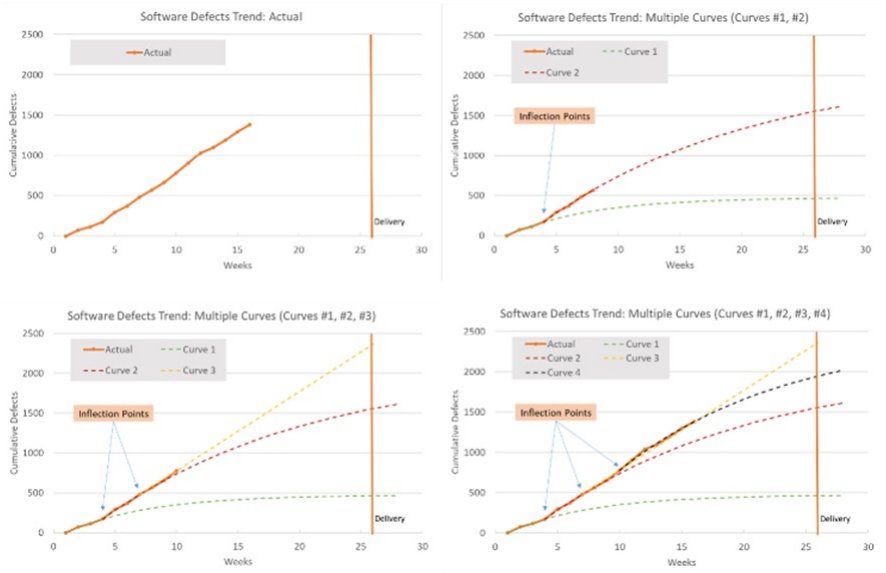

Fig. 6: An illustration of the construction process for generating a series of piece-wise exponential curves with inflection points

$$A_i = (y_i - y_{i-1})\{x_i \exp(-bx_i) - x_{i-1} \exp(-bx_{i-1})\} \tag{7}$$

$$B_i = \exp(-bx_{i-1}) - \exp(-bx_i) \tag{8}$$

$$C = \frac{x_p y_p}{\exp(bx_p) - 1}. \tag{9}$$

To obtain the maximum likelihood estimates of a and b, equations (5) and (6) can be solved numerically. We have developed an algorithm in STAR capable of automatically solving these equations. It's important to note that a predicted curve derived from the maximum likelihood estimates of a and b always intersects the initial data point (x_0, y_0) and the final data point (x_p, y_p). In the next section, we will employ this estimation method to generate multiple exponential curves.

3.1.2 A piece-wise exponential model

Now, we will introduce a multi-curve methodology, particularly focusing on employing multiple exponential curves, each representing distinct stages of the development process, as illustrated in Fig 6, $n(t) = m(t - \alpha)$.

The underlying concept here is that not all features are available for testing at once; instead, subsets become accessible gradually over time. Consequently, this approach results in multiple waves of defects emerging during the testing phase.

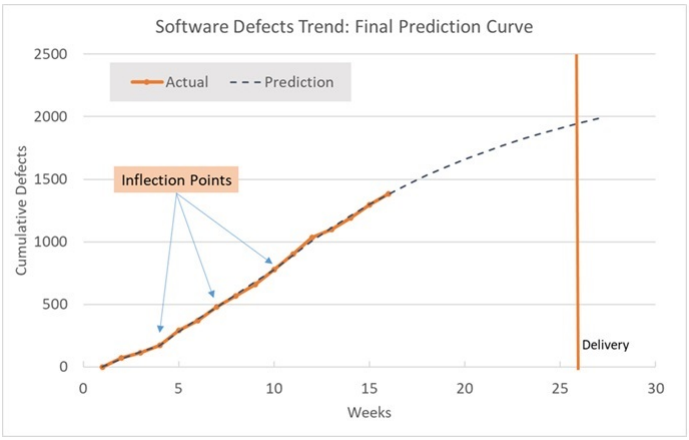

Fig. 7: A final defect arrival curve using the multiple curve approach

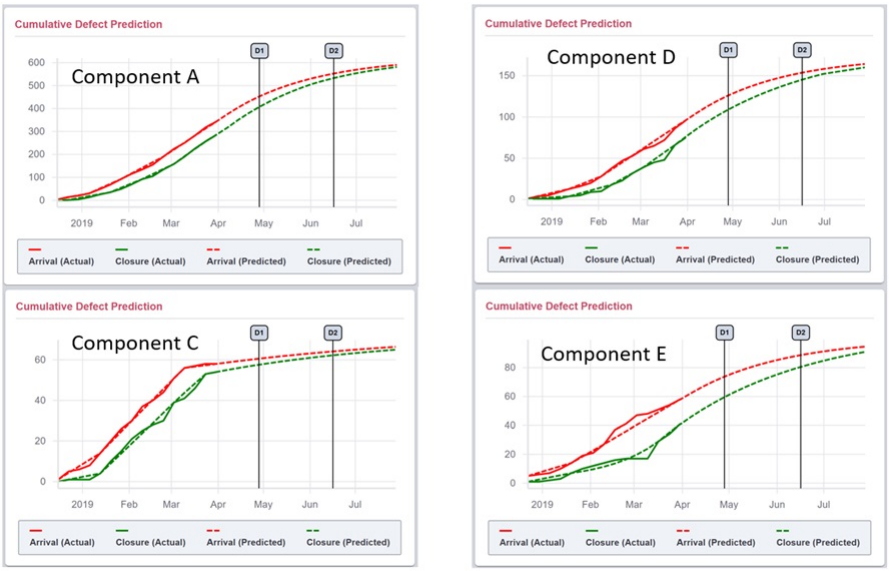

Fig. 8: Closure prediction from arrival predicted curve

An algorithm [19] has been developed to automatically identify inflection points, indicating significant shifts in trends, and apply a statistical parameter estimation technique. The outcome is an exceptionally accurate fit for the data, as depicted in Fig 7. Using a single curve approach would not suffice for conducting the defect trend

analysis. Advancements in computing technology have facilitated the computation of intricate data.

Mathematically, the resulting piece-wise exponential curve is represented as follows: For period i, we have

$$m_i(t) = a_i\{1 - \exp[-b_i(t - t_{i-1})]\} + m_{i-1}(t_{i-1}) \quad \textbf{for} \quad t_{i-1} < t < t_i \tag{10}$$

It's important to note that $m_i(t)$ signifies the cumulative defects detected by time t, situated between the two inflection points, t_{i-1}, and t_i, where t falls within this time frame. The parameters a_i, and b_i represent the total defects and defect rate, respectively, for period i. These values can be determined utilizing the statistical method of maximum likelihood discussed in Sect.3.1.1.

Furthermore, this algorithm offers flexibility to accommodate various sizes and patterns of defect data. Fig 8 demonstrates the algorithm's adaptability in four distinct scenarios, where the actual and predicted arrival curves are indicated in red. The green curves represent actual defect closure data with solid lines and predicted closure curves with dashed lines. The closure prediction method will be discussed in the upcoming section.

3.2 Closure curve prediction

The input data comprises the leading predicted arrival curve and the actual closure data, as illustrated in Fig 9. Through thorough analysis of extensive project data, we have established that the predicted closure curve can be generated by shifting the leading data to the right. If we represent the cumulative predicted arrival curve over time t by $m(t)$, the cumulative closure predicted curve, $n(t)$, is derived by shifting $m(t)$ by a constant value α, in the following manner:

$$n(t) = m(t - \alpha) \tag{11}$$

To achieve the most accurate fitting curve, multiple real closure data points are utilized. The default setting for this analysis is 2, meaning that the last 2 data points are taken into account. In this particular instance, through numerical calculations, we determined the optimal value for α to be 2.3. This value was obtained to minimize the sum of squares of differences between the predicted curve and the actual closure data points. In this example, the closure prediction is achieved by shifting the arrival curve by 2.3 weeks. Fig 9 showcases the predicted closure curve alongside the actual data, visually confirming the accuracy of the algorithm.

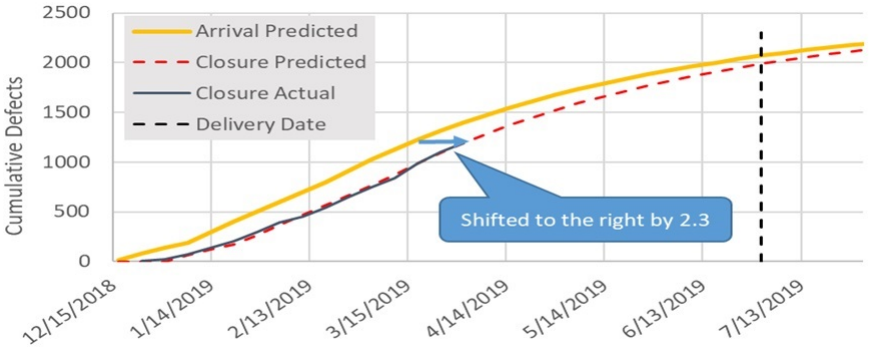

Fig. 9: Closure prediction from arrival predicted curve

3.3 Defect open curve prediction

In practical situations, one of the key metrics is the count of open defects, commonly referred to as backlog defects. This metric signifies defects that are yet to be resolved or fixed. Ideally, all identified defects should be rectified before the software is delivered. The open defect curve can be derived by computing the difference between the arrival curve and the closure curve, expressed as:

$$o(t) = m(t) - n(t). \tag{12}$$

Utilizing the dataset discussed in this section, we calculated the open defect curves for both actual and predicted values using equation (12). Fig 10 demonstrates the close alignment between the actual data and the predicted curve.

3.4 Early defect prediction

Conventional SRGMs, as discussed in Sect.3.1, rely on defect data collected during the software testing phase. This limitation poses significant challenges, especially in the early stages of software development when crucial decisions like staffing levels, required testing efforts, or focus on specific features must be made. These decisions are not only important but also time-consuming. With the industry trend shifting towards extremely short software development cycles, as seen in agile development, making accurate decisions early in the development phase is critical.

Especially in the initial planning phase, projects often require insights into how the defect-finding curve might appear during internal testing. This understanding is vital for determining the necessary staffing levels for development and testing activities. Consequently, predicting defects early in the software development pro-

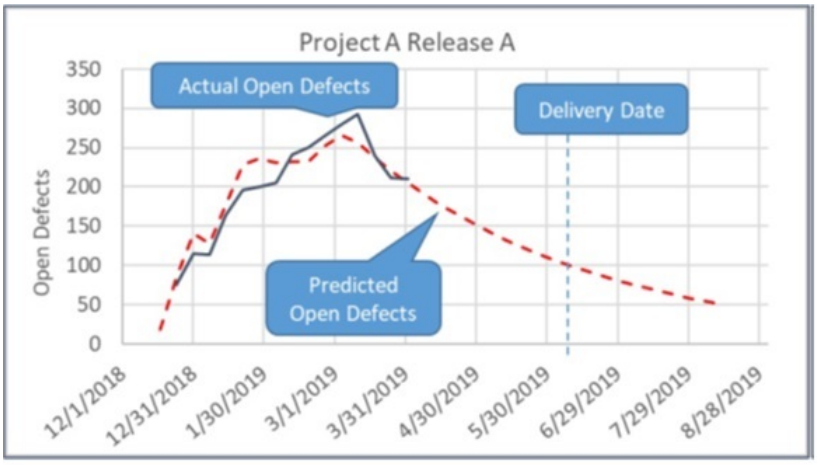

Fig. 10: Defect open predicted curve

cess becomes essential. This early prediction, referred to as early defect prediction, holds significant value, particularly when actual defect data is unavailable. It aids in identifying software quality issues, preventing cost overruns, and establishing an optimal development strategy.

3.4.1 Input data

We are introducing the concept of leading data to enhance early defect prediction. In recent years, we explored potential leading data based on its availability and correlation with defects. Our research led us to identify effort data, specifically in two forms (development and test), as the most suitable for this purpose. During the planning phase, these two types of effort data are easily accessible and are typically measured in hours required for completing specific development activities, often at a sub-feature level.

- Development effort data: This data indicates the complexity of the software content. The development effort curve demonstrates how the content evolves over time and is crucial in predicting the number of defects.

- Test effort data: This data illustrates testing progress when cumulative test effort is standardized. The rate at which defects are detected is closely linked to the test progress. Essentially, the shape of the defect discovery curve mirrors the test progress curve.

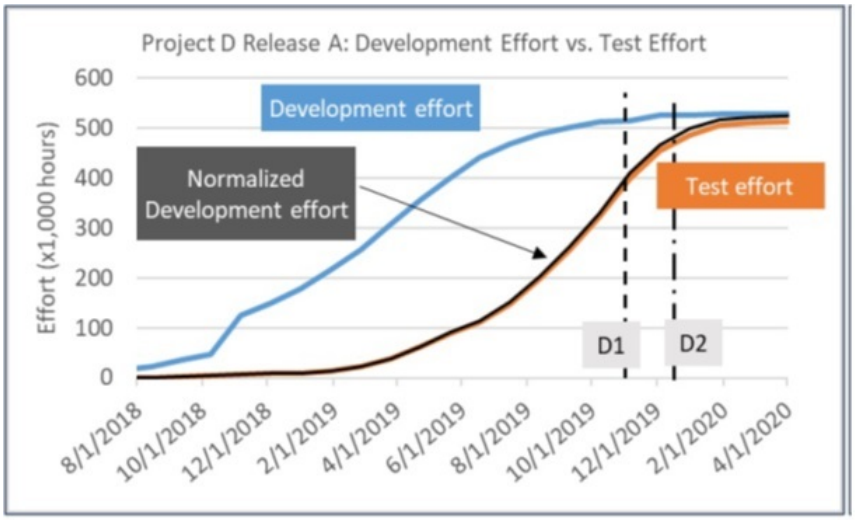

Fig. 11: Effort data: Development, test, normalized development

Fig 11 presents a depiction of standard cumulative development and test effort data. It's crucial to observe the notable disparity between the development and test effort curves. In extensive development projects, integrating numerous low-level modules is necessary for a feature to be prepared for testing. Defects are typically identified during feature-level tests, underscoring the discrepancies between development and test efforts.

3.4.2 Preprocessing effort data

Initially, effort data suitable for the prediction algorithm is prepared by merging development and test efforts to create the leading data.

- Normalizing development effort data by test progress:

  Development effort data is normalized by the test progress, calculated as the test effort curve divided by the maximum value of the test effort data. This normalized development effort curve illustrates how the software content is tested over time (shown in Fig 11).

- Adjusting the tail end of the normalized development effort curve:

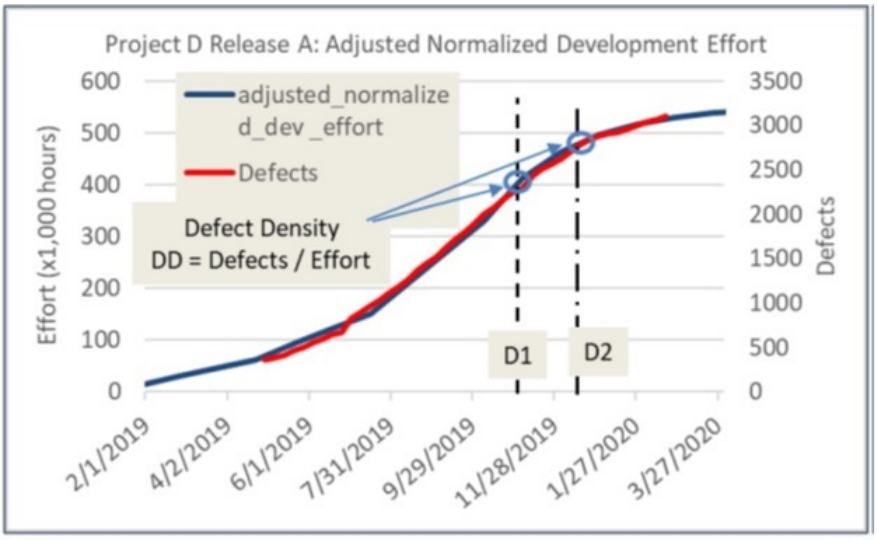

Fig. 12: Normalized development effort curve with the adjustment

Modification of the tail end of the normalized curve becomes necessary due to its rapid flattening trend. This occurs because the test effort covers internal testing activities, while defect data usually includes issues identified during customer testing and operation. To accommodate customer-related defects, an adjustment is made, utilizing the trend analysis algorithm detailed in Sect.3.1.1.

- Calculation of defect density:

  Defect densities at $D1$ and $D2$ are computed using the defect curve and the adjusted normalized effort curve. Utilizing an interpolation technique, values at $D1$ and $D2$ are determined. Defect density is then calculated as defects over effort hours. Fig 12 illustrates the adjusted normalized effort curve overlaid with the defect arrival curve, emphasizing the close association between the defect and effort curves.

- Target defect values at the delivery date:

  The same procedure is applied to the effort data from the current release to establish the adjusted normal development effort curve. Using the calculated defect densities, target defects at $D1$ and $D2$ for the ongoing release are determined by multiplying defect density with the effort hours of the current release. This results in our input data, comprising the effort curve and the target defects

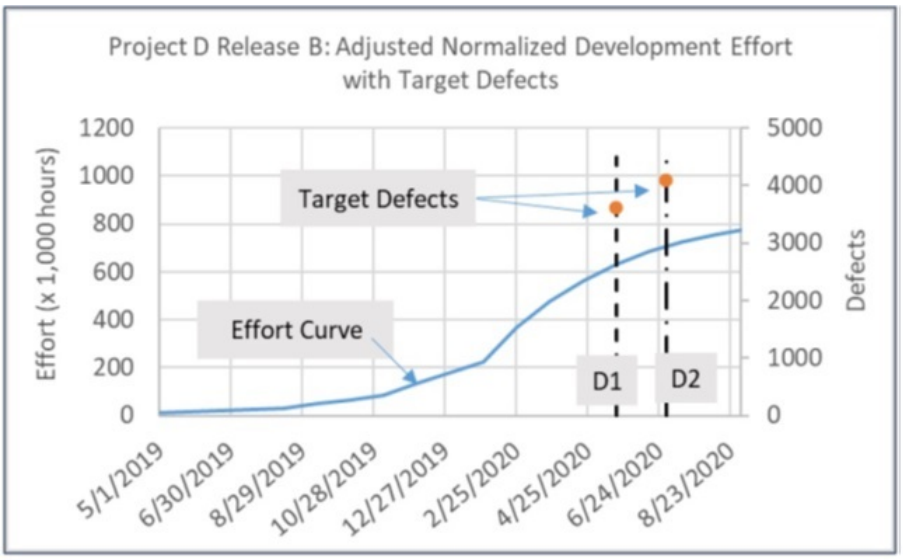

Fig. 13: Target defects vs. adjusted normalized development effort curve

at $D1$ and $D2$, as depicted in Fig 13

3.4.3 Transformation Algorithm for Early Defect Prediction

The transformation function consists of two elements: horizontal and vertical shifts. In simpler terms, the original curve, referred to as the effort curve, undergoes a transformation involving both horizontal and vertical shifts to closely match the target defects at $D1$ and $D2$. The transformation function, which maps (x, y) coordinates to (x_{new}, y_{new}) is detailed in equation (13) for horizontal and vertical shifts as respectively,

$$x_{new} = \alpha + \beta x, \quad y_{new} = \gamma y. \tag{13}$$

The parameter α signifies a fixed delay in weeks from the effort curve to the defect curve, while β represents an additional delay in the defect curve. The parameter γ is determined as the ratio of defects to effort hours, indicating defects per effort hour. This problem involves non-linear optimization with three variables (or parameters) and aims to minimize the total distance between predicted defects and target defects at $D1$ and $D2$ using two data points. Solving this problem requires iterative numerical analysis involving three-level loops and multiple iterations for each level. Initial

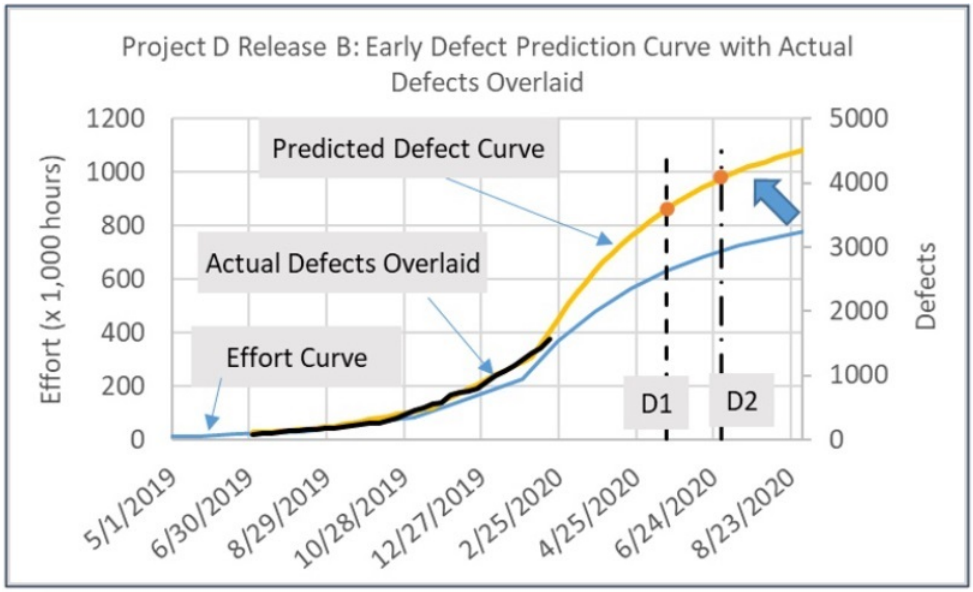

Fig. 14: Early defect prediction with actual defects overlaid

values, increments, and maximum iteration counts must be defined for each level. It's important to note that the defect closure prediction discussed in Sect.3.2 is a particular case where $\beta = 1$ and $\gamma = 1$.

Fig 14 showcases the ultimate defect arrival curve, which is expected to closely match the target defects at D1 and D2. To validate the early defect prediction, actual defect data can be superimposed, as shown in the same figure.

4 Use cases showcasing the capabilities of STAR's core engine

We will now explore various commonly asked questions by employing the techniques implemented in STAR's core engine, as detailed in Sect.3.

4.1 Use Case #1: Will we find enough defects?

We normalize residual defects by the total number of defects, making this metric applicable to a wide range of projects. The definition is as follows:

$$\textbf{\% Residual Defects} = \frac{\textbf{Total defects} - \textbf{Defects found}}{\textbf{Total defects}} \tag{14}$$

This metric evaluates the probability of detecting an adequate number of defects. Our recommended threshold values, derived from practical experience, are as follows: It is deemed acceptable (green) if it's less than or equal to 15%, at risk (red) if it exceeds 25%, and a cautionary zone (yellow) for values falling in between.

STAR autonomously generates an arrival curve using the techniques outlined in Sect.3 and computes the percentage of residual defects. The resulting visualization is presented in Fig 15. The upper table displays the defect counts at significant milestones for the arrival, closure, and open curves. Specifically, the defect counts linked to the arrival curve will be employed to compute the percentage of residual defects at D1 and D2, as per equation (14). The outcomes are presented in the table located in the middle left section.

4.2 Use Case #2: Will we fix enough defects?

Open defects are standardized by the total number of defects discovered, represented as:

$$\textbf{\% Open defects} = \frac{\textbf{Defects found} - \textbf{Defects fixed}}{\textbf{Defects found}} \tag{15}$$

This metric assesses if a sufficient number of defects will be resolved. Based on practical experience, our suggested threshold values are as follows: It is considered acceptable (green) if it's less than or equal to 5%, at risk (red) if it exceeds 10%, and a cautionary zone (yellow) for values falling in between. STAR autonomously generates an open curve using the techniques outlined in Sect.3 and computes the percentage of open defects. The resulting visualization is presented in Fig 15. The defect counts linked to the arrival and closure curves in the upper table will be employed to compute the percentage of open defects at $D1$ and$D2$, as per equation (15). The outcomes are presented in the table located in the middle right section. The 90% lower and upper confidence limits are also added to the chart.

Fig. 15: Percentages of residual and open defects

4.3 Use Case #3: Will we be ready for delivery on time with acceptable quality?

By incorporating the quality metrics discussed in Sects.4.1 and 4.2, we are now equipped to make well-informed decisions about the software release's preparedness. The color-coding scheme of green, yellow, and red corresponds to 'Acceptable,' 'Warning,' and 'At risk,' respectively. In this instance, we conclude that the release will be ready for deployment according to the desired quality at D2 (Delivery for Deployment) but not at D1 (Delivery for Trial). Table 1 provides an example of software release readiness metrics.

4.4 Use Case #4: What if we don't have defect data early in development? How can we make early quality estimates?

The dataset from Project D is employed to showcase early defect prediction for Release B, utilizing defect and effort data from the preceding release, Release A. As

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%

Table 1: A sampe of key quality metrics for the release realines evaluation

Fig. 16: Percentages of residual and open defects

explained in Sect.3.4, STAR utilizes an innovative method to transform the effort data (as leading data) into a defect curve. Fig 16 illustrates the effort curve transformed into the predicted curve, which aligns with the target values at $D1$ and $D2$. This represents an early defect prediction without actual defect data during the planning phase. To demonstrate this prediction, we overlaid actual defect data from Release B, depicted in Fig 17. The fit is remarkably accurate.

4.5 Use Case #5: What can we do to improve the software quality? And what if we increased the number of developers or delayed the delivery?

A common question in software quality assurance is: How can we improve software quality, and to what extent can corrective measures enhance it? STAR offers real-time quantification of the quality impact of corrective actions. We demonstrate this

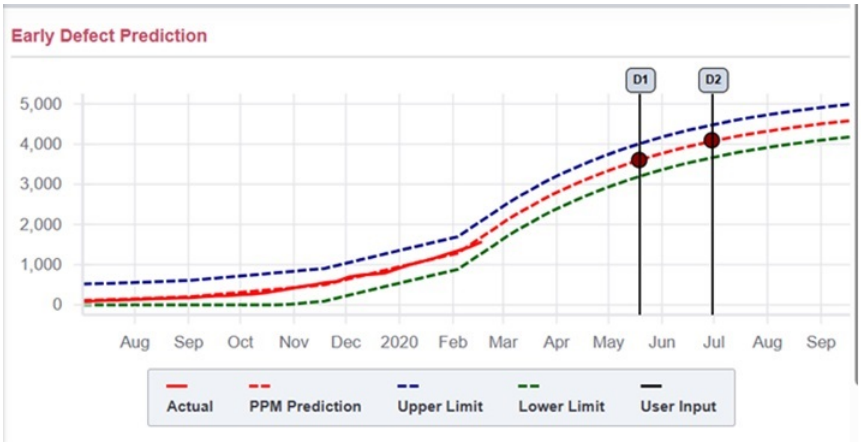

Fig. 17: Percentages of residual and open defects

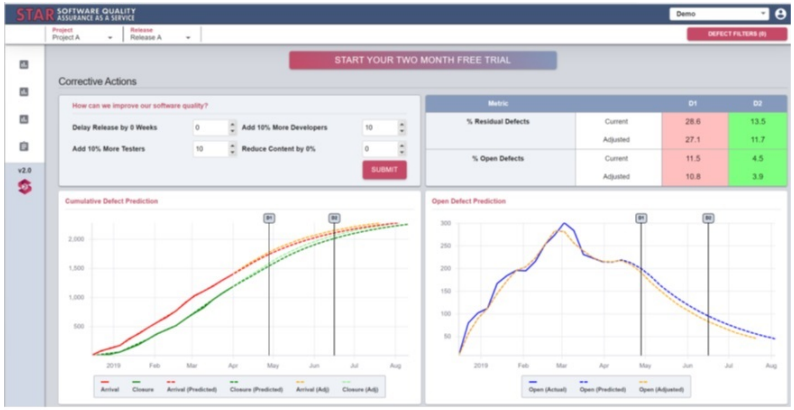

Fig. 18: A sample output of the STAR corrective actions page

interactive approach for commonly employed corrective actions: 1) Delaying the delivery, 2) Increasing the number of developers, 3) Expanding the team of testers, and 4) Reducing the release content, i.e., decreasing the number of features. STAR allows for various combinations of these options. In Fig 18, we present a scenario involving two choices: 'Increase testers by 10%' and 'Add 10% more developers.' The metrics table shows updated values for % Residual and % Open at $D1$ and $D2$. Both metrics indicate improvement. The charts display adjusted arrival, closure, and open curves based on these options.

Fig. 19: Prediction stability based on the multiple exponential curves method

4.6 Use Case #6: When can we start using the prediction? Are predicted values stable and accurate?

Many SRGMs employ goodness-of-fit as a measure for model validation. However, it is crucial to assess the stability and accuracy of predictions in real-world applications. We iterate the process outlined in Sect.3.1 as fresh weekly data becomes available. Every week, we track the projected defects at the delivery date, $D2$. The results summary, shown in Fig 19, includes the average predicted defects at $D2$ with +/- 10% margins. Despite near-perfect goodness-of-fit, the prediction's stability remains questionable. Typically, stability is achieved several weeks before the delivery date, aligning with the start of system testing. During this period, the trend analysis results cannot be fully relied upon for decision-making.

We've introduced an additional feature to the early defect prediction model, incorporating actual defect data as it becomes accessible. This inclusion provides prediction stability data, as depicted in Fig 20. This data exhibits a remarkably consistent trend from the early stages of the test phase, maintaining accuracy within the range of less than +/- 10%. This suggests that early defect prediction not only offers a highly stable and precise forecast but also demonstrates exceptional goodness-of-fit, extending up to the current week. This emphasizes the effectiveness of utilizing development and test effort data along with data from previous releases, enhancing both prediction stability and accuracy.

It's crucial to note that many studies comparing SRGMs often rely solely on goodness-of-fit, which may not be a fair comparison criterion due to the dynamic nature of the defect detection process over time.

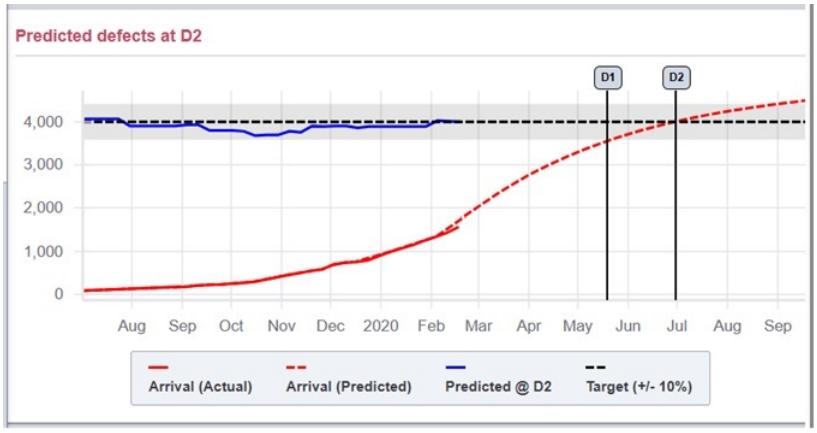

Fig. 20: An early defect prediction model with actual defect data

Fig. 21: Defect breakdown by component

4.7 Which of our software components exhibits the highest defect rate?

Examining the breakdown of the defect trend by component helps identify problematic components while analyzing the breakdown by severity focuses attention on critical and major defects. Refer to Fig 21 for an example output illustrating defects categorized by component.

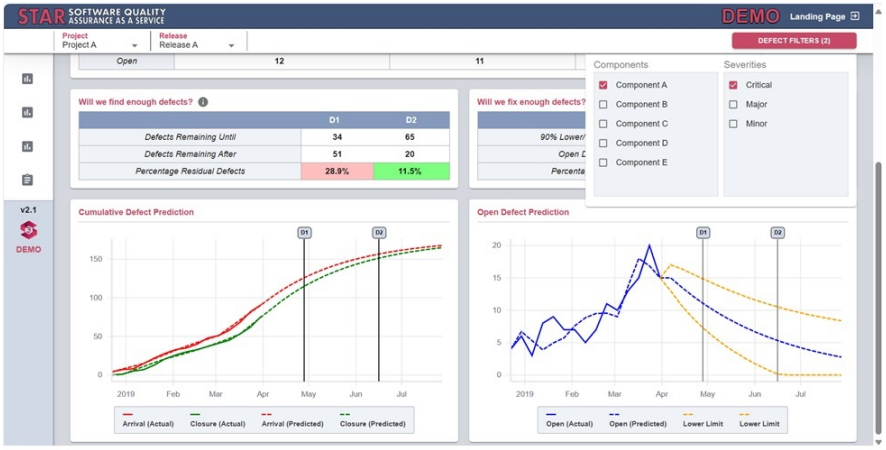

Fig. 22: Defect breakdown by component

The defect filters allow users to select specific components and severity levels, showcasing the versatility of automatic prediction. Fig 22 illustrates these defect filters, and additional customization options, including component and severity filters, cater to a broad audience, ranging from developers to executives. It's essential to note that the defect filter was utilized to create the specific visualization shown in Fig 8 in Sect.3.1.

This functionality also facilitates exploring detailed breakdowns of defects, such as distinguishing software from hardware issues, differentiating cybersecurity from non-cybersecurity concerns, comparing internal tests with customer tests and operations, and identifying unique faults versus duplicates or non-issues within various application domains.

5 Conclusions

We've introduced a groundbreaking software quality assurance tool called STAR. It not only addresses common concerns but also aids project managers in making well-informed decisions about development resources and release quality. STAR offers an intuitive interface paired with advanced visualization. Its core engine incorporates robust statistical analytics, extensively tested with real project data, ensuring stable and precise defect predictions. STAR utilizes a series of piece-wise exponential models and leverages development and test effort data from previous releases for prediction stability and accuracy. Compared to traditional single-curve fitting mod-

els, STAR exhibits outstanding goodness of fit. As a cloud-based tool, it is accessible anytime, anywhere, and by anyone, making it publicly available for use.

Acknowledgements I would like to thank Dr. Rashid Mijumbi, Rory Harpur, Joseph Good, and Michael Okumoto for their significant contributions.

References

1. Software as a service, https://en.wikipedia.org/wiki/Software_as_a_service.
2. Lionel Sujay Vailshery (2023) Size of the Software as a Service (SaaS) market worldwide.
3. H. Okamura and T. Dohi (2021) 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
4. A. Nikora, L. Fiondella and T. Wandji (2018) SFRAT- An Extendable Software Reliability Assessment Tool, IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2018, pp. 65-70, doi: 10.1109/ISSREW.2018.00-29.
5. M. R. Lyu (1995) Handbook of Software Reliability Engineering, Computer Society Press, Los Alamitos, and McGraw-Hill, New York.
6. List of software reliability models, Wikipedia, https://en.wikipedia.org/wiki/List_of_software_reliability_models.
7. CASRE(1992) a computer-aided software reliability estimation tool. doi:10.1109/CASE.1992.200165.
8. Musa JD (1975) A theory of software engineering and its application. IEEE Transactions on Software Engineering, SE-1. 3:312-327.
9. H. Pham (1999) Software Reliability. Berlin, Heidelberg: Springer-Verlag.
10. S. Yamada, M. Ohba, and S. Osaki (1983) S-Shaped Reliability Growth Modeling for Software Error Detection, IEEE Trans Reliab, vol. R-32, no. 5, pp. 475-484,doi: 10.1109/TR.1983.5221735.
11. H. Pham, L. Nordmann, and X. Zhang (1999) A general imperfect-software-debugging model with s-shaped fault-detection rate, IEEE Trans Reliab, vol. 48, no. 2, doi: 10.1109/24.784276.
12. D. R. Jeske, X. Zhang, and L. Pham (2005) Adjusting software failure rates that are estimated from test data, IEEE Trans Reliab, vol.54, no.1, doi: 10.1109/TR.2004.842531.
13. J. Sorrentino, P. Silva, G. Baye, G. Kul, and L. Fiondella (2022) Covariate Software Vulnerability Discovery Model to Support Cybersecurity Test Evaluation (Practical Experience Report), 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), pp. 157-168. doi: 10.1109/ISSRE55969.2022.00025.
14. k. Okumoto (2017) An Overview of Practical Software Reliability Prediction, Chapter in Reliability Modeling with Computer and Maintenance Applications, edited by Nakamura et al, World Scientific Publishing Co., pp. 3-21.
15. A. L. Goel and K. Okumoto (1979) Time-dependent error-detection rate model for software reliability and other performance measures, IEEE Transactions on Reliability, pp. 206-211.
16. J. D. Musa, A. Iannino, and K. Okumoto (1987) Software Reliability: Measurement, Prediction, Application. USA: McGraw-Hill, Inc.,.
17. R. Mijumbi, K. Okumoto, A. Asthana, J. Meekel (2018) Recent Advances in Software Quality Assurance, IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Oct 2018, pp. 77-82, Memphis, TN, USA.
18. K. Okumoto, R. Mijumbi, A. Asthana (2018) Software Quality Assurance, Chapter in Telecommunication Networks: Trends and Developments, Intech Open, Nov 2018. [Online]. Available: https://www.intechopen.com/download/

19. K. Okumoto (2022) Early Software Defect Prediction: Right-Shifting Software Effort Data into a Defect Curve, 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 43?48. doi: 10.1109/ISSREW55968.2022.00037.
20. K. Okumoto(2023) Digital Engineering-driven Software Quality Assurance-as-a-Service, in Proceedings - 29th ISSAT International Conference on Reliability and Quality in Design.