Next-Generation Software Reliability Assessment Using a Cloud-Based Tool

Kazuhira Okumoto, PhD, Sakura Software Solutions, LLC

Keywords: Software reliability, defect prediction, cloud analytics, quality assurance, software engineering

SUMMARY & CONCLUSIONS

This paper introduces STAR, a next-generation, cloud-based tool that transforms software reliability assessment through data-driven prediction and decision support. Unlike traditional single-curve software reliability growth models (SRGMs), STAR employs a flexible multi-curve algorithm that adapts to the dynamics of modern software development, including agile practices, staggered feature releases, and variable defect discovery rates.

STAR enables project managers, developers, and quality assurance teams to make informed decisions throughout the software development lifecycle. By integrating real project data, STAR provides:

- Accurate predictions of defect arrival, closure, and open backlog
- Key quality metrics such as %Residual and %Open defects
- Visualizations that make complex statistical results easily interpretable
- Early defect prediction using effort-based leading indicators
- Simulations of corrective actions and their real-time impact on quality

Through its intuitive web-based interface, STAR ensures global teams' accessibility at any time. It eliminates the need for specialized statistical expertise or local installations, making high-end software quality analytics available as a Software-as-a-Service (SaaS) platform. STAR has successfully applied its technology in industrial and research settings, including deployment in Nokia's flagship division, a current trial collaboration with NASA, and an extended trial with System Engineering Consultant (SEC).

Several enhancements are planned for STAR. These include:

- Integration of AI to support adaptive learning in defect prediction and automated anomaly detection
- Incorporation of cybersecurity factors, enabling STAR to assess vulnerabilities in defect data related to security
- Expansion into hardware reliability analysis by integrating STAR with FUSION, an NSF-funded digital engineering platform, enabling concurrent evaluation of hardware and software failures

A novel prediction technique based on historical release data, akin to machine learning for dynamic environments, further improves early-stage forecasting. Together, these advancements position STAR as a powerful, extensible platform capable of evolving with the needs of modern software and system engineering organizations.

1. INTRODUCTION AND BACKGROUND

Digital transformation [1] involves adopting digital technologies to improve or reinvent products, services, and operations, ultimately enhancing value through innovation and efficiency. At its core, cloud technologies enable agility, collaboration, and customer focus. Imagine a future where quality teams no longer manage custom spreadsheets. Instead, data from defect logging systems is automatically collected, cleaned, and transformed into reliability analytics. Quality managers can access real-time metrics to guide launch decisions without relying on technical experts or complex models. STAR, the tool introduced in this paper, enables this future, empowering teams to make data-driven decisions with ease and precision.

The software reliability market is undergoing a significant shift, driven by growing demand for specialized tools. The software quality assurance sector is expected to reach $20.8 billion by 2030, growing at a CAGR of 8.8% [2]. This paper introduces STAR—a cloud-based software reliability tool that combines advanced analytics and visualization to support users across all roles in understanding and improving software quality.

a. Software Development Process vs. Defect Injection & Removal

A software release consists of newly developed features or functionalities that go through a structured software development lifecycle, including requirements specification, software design, coding, and testing against predefined criteria. This lifecycle is illustrated in Figure 1. Upon completing internal testing, the software enters the acceptance testing phase at the customer site before final deployment for commercial use. Software defects are identified during both internal and customer testing. Quality improves over time through an iterative "find-and-fix" process that includes internal testing, customer testing, and

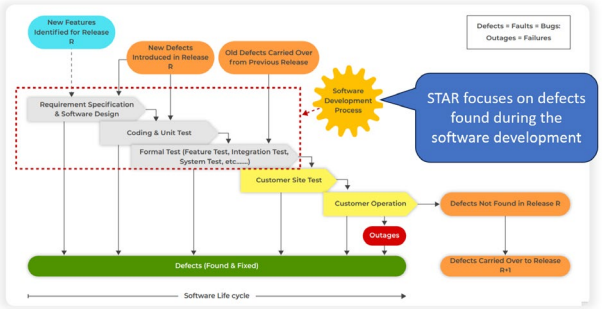

FIGURE 1. SOFTWARE DEVELOPMENT PROCESS VS. DEFECT INJECTION AND REMOVAL

operational feedback. As a result, development teams often address defects discovered internally and externally after the initial software delivery.

Nevertheless, some defects surface during the operational phase after deployment, potentially leading to service interruptions or system failures. Understanding the distinction between defects and failures is essential. In this study, we classify critical severity defects as software failures, based on standard severity classifications. This paper focuses on defects found during software development and testing.

b. *Software Reliability Growth Modeling (SRGM)*

A primary challenge engineering teams face is evaluating software quality and using that evaluation to guide release decisions. Predicting the number of remaining defects and their potential impact is particularly critical in projects with fixed delivery schedules.

Software Reliability Growth Models (SRGMs) describe how the reliability of software improves as defects are discovered and corrected during testing or operation. Reliability is not static; it evolves as failures are observed and fixes are applied. SRGMs provide a quantitative framework to measure, predict, and manage this improvement.

These models analyze cumulative defect detection data to identify the trend or shape of the defect curve. Early work on SRGMs in the 1970s [3–5] began with simple representations, such as step functions for defect rates, a significant advancement at the time. Later, the models evolved into stochastic processes, most notably exponential growth curves [5], where defect discovery was modeled as a time-dependent process. Applied initially to system test data, these models proved reliable and gained broad adoption, encouraging researchers to extend their applicability to broader testing phases.

As software testing became more complex, modeling defect discovery across the entire test lifecycle grew increasingly complex. Over 200 SRGMs have since been proposed [6–14], developed under various assumptions, and tailored to different testing environments. Most of these models rely on a single, continuous curve to describe the defect discovery process throughout testing.

We selected the top five SRGMs based on their high citation frequency, broad industrial adoption, historical significance, and diversity of modeling approaches. Table I summarizes each model with its year of introduction, type, underlying assumptions, key equations, and representative applications.

Model	Year	Type	Assumptions	Key Equation	Applications
Jelinski–Moranda	1972	Execution-time	Fixed faults; equal hazard; failure rate drops as faults are removed	$\lambda(t) \propto$ remaining faults	Foundational, teaching, benchmark
Littlewood–Verrall	1973	Bayesian	Prior on failure intensity; update with data	Posterior $\propto$ prior × likelihood	Limited data; parameter uncertainty
Goel–Okumoto	1979	NHPP (Exponential)	Poisson arrivals; exponential decay in rate	$m(t) = a\,(1 - \exp(-bt))$	Widely used baseline, industry
Yamada S-Shaped	1983	NHPP (S-shaped)	Learning effects: slow-then-fast-then-saturate	$m(t) = a\,(1 - (1+bt)\exp(-bt))$	Projects with delayed detection
Musa–Okumoto	1984	NHPP (Logarithmic)	Logarithmic decay in the execution time domain	$m(\tau) = (1/\theta)\ln(1+\theta N\tau)$	Industrial use, Bell Labs, NASA

TABLE 1. COMPARISON OF TOP 5 SRGMS

c. *Non-Homogeneous Poisson Process (NHPP)*

Most SRGMs are formulated using the concept of the Non-Homogeneous Poisson Process (NHPP). To conceptualize this, consider a finite number a of defects. The probability that a defect is discovered by time t follows a cumulative distribution function $F(t)$. The defect discovery process $N(t)$ can initially be modeled using a binomial distribution, as shown in equation (1).

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

Since a is typically large, the binomial distribution is approximated by a Poisson distribution with a mean value function $m(t)$, as shown in equation (2).

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

FIGURE 2. SAMPLE SOFTWARE RELIABILITY GROWTH MODEL (SRGM)

This yields a generalized NHPP model. It implies that the number of defects found in a given time interval follows a Poisson distribution with a mean based on the interval's length. The mean value function m(t) = $aF(t)$ represents the expected number of defects detected by time t. For example,

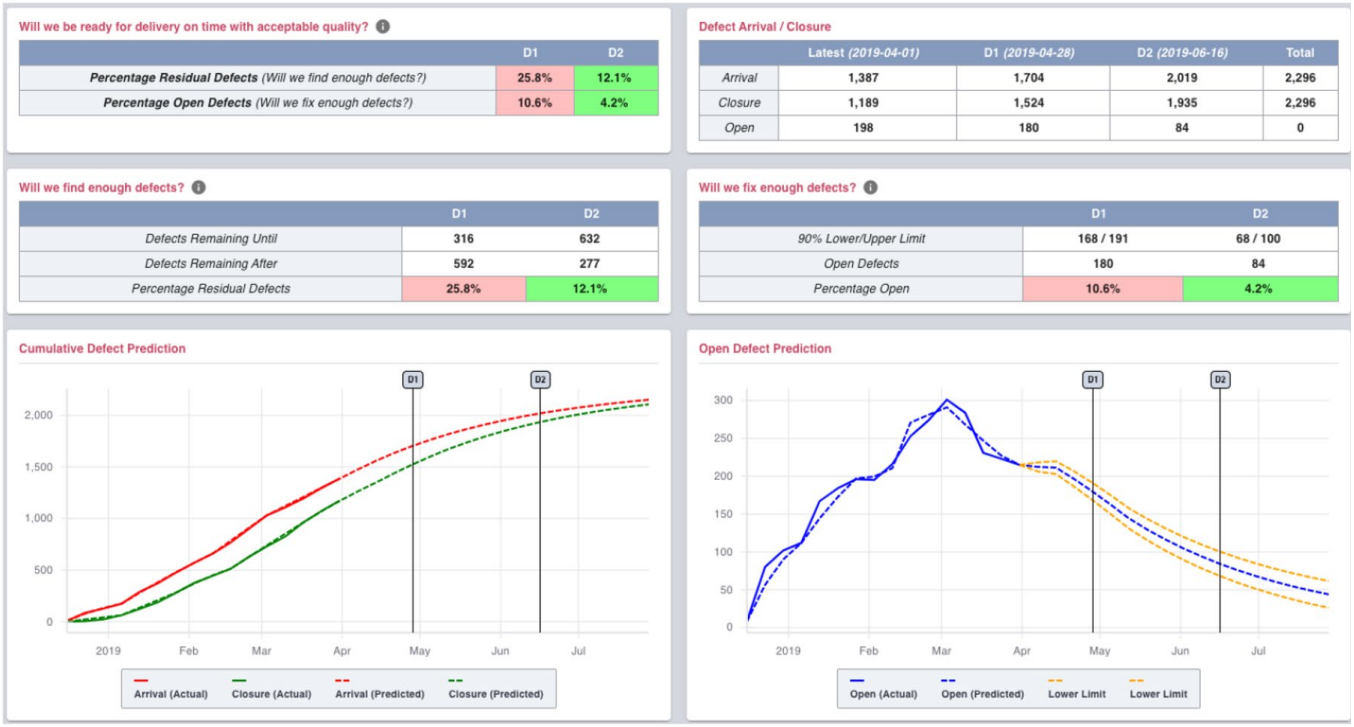

FIGURE 3. STAR EXECUTIVE SUMMARY VIEW

the exponential model is described using NHPP, where the mean value function has the form shown in equation (3).

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

The NHPP assumption enables statistical methods, such as maximum likelihood estimation (MLE), to determine parameters a and b, as well as confidence intervals, using standard approximation techniques.

Consider the following example in Figure 2: Suppose an SRGM forecasts 4500 defects. At week 19 (the current time), 2500 defects have been found. Assuming testing continues steadily, the model predicts an additional 1200 defects will be discovered over the next 11 weeks before release. Thus, the number of remaining defects at delivery is 800 (= 4500 – 3700), resulting in a residual defect percentage of approximately 18% (= 800 / 4500).

These metrics are crucial for informed decision-making and will be further explored in Section 4.

d. State-of-the-Art in SRGMs

Exponential SRGMs [3, 4, 5] introduced more than 40 years ago remain widely used due to their simplicity and clarity. They assume that the software contains a finite number of defects, each of which is independently discovered over time according to an exponential distribution. These models utilize the NHPP framework to statistically estimate key parameters, forming the basis for various software quality metrics. While initially applied to system testing, exponential models have since been adapted to span the entire testing lifecycle, including feature, integration, and functional testing.

Newer models introduce additional complexity to enhance adaptability by incorporating alternative statistical distributions, such as the Weibull, Gamma, and logistic distributions. These modifications enable more accurate modeling of real-world defect discovery patterns, particularly when the data indicates fluctuating detection rates. Such models often result in S-shaped curves, as described in references [6-9, 11-13].

2. STAR OVERVIEW

a. What is STAR?

STAR [15, 16] is a transformative digital engineering tool designed to modernize software quality assurance by integrating data-driven analysis into the software release decision-making process. This advanced, cloud-based platform enables real-time evaluation of software quality. It is developed explicitly for practitioners responsible for ensuring product quality and making critical decisions about software readiness for release. Being web-based and fully automated, STAR promotes seamless collaboration across different teams, projects, and product components.

STAR combines the principles of Software as a Service (SaaS), automated analytics, and dynamic visualizations to address many of the questions and challenges. As a SaaS solution, STAR enables users to access the platform from any location, eliminating the need for software installation or maintenance, and significantly improving usability and deployment speed. By embedding automated analytics, STAR removes the need for specialized domain expertise, making advanced statistical modeling accessible to a broader range of users. Its visualization capabilities simplify complex results, allowing stakeholders to quickly interpret trends and insights, thereby supporting faster, evidence-based decision-making.

Figure 3 showcases STAR outputs, including predicted and actual defect arrival, closure, and open curves. The top-left table shows the current quality snapshot at two key milestones: D1 (Delivery for Trial) and D2 (Delivery for Deployment). This enables project managers to assess whether the software meets the defined quality thresholds for each stage. Additional tables provide supportive quantitative data, while the charts visually compare actual trends with predicted ones. Analytics behind the charts will be discussed in Section 4.

STAR also addresses the broader challenges raised in earlier sections, especially those related to accessibility, collaboration, and technical barriers. A previous version of STAR [17-19] was tailored and deployed in Nokia's leading business unit, where it played a key role in supporting their Quality Improvement Process (QIP). The current version is also undergoing a trial collaboration with NASA, further validating its applicability in complex, high-stakes environments.

b. STAR ARCHITECTURE

STAR streamlines the entire data processing pipeline—from raw input acquisition to final analytics output. Its architecture, illustrated in Figure 4, is deployed on the AWS cloud platform and designed to handle scalability, performance, and interoperability.

The process begins with data extraction from various defect tracking systems via RESTful APIs implemented in Flask. Extracted data is stored in two separate databases: PostgreSQL for structured data and DynamoDB (a NoSQL database) for unstructured or semi-structured data. This dual-database approach enhances performance and scalability across queries and analytics. Before storage, a comprehensive pre-processing phase ensures that the data is consistently formatted and prepared. This step includes time-based aggregation (e.g., weekly bins) and alignment of input variables for the analytics engine. A crucial pre-processing function is to harmonize inconsistencies across multiple sources and projects. For example, different systems might use labels for the same attribute, such as 'priority' versus 'severity'. STAR resolves these discrepancies through intelligent field mapping and normalization.

Pre-processing also handles contextual interpretation, such as mapping a defect to a specific geographic region or organizational unit, and detecting duplicates or cross-release redundancies. These functions ensure that quality assessments and cross-project comparisons remain valid and meaningful. The core analytics engine is built in Python, enabling advanced statistical modeling and machine learning capabilities. The user interface uses JavaScript ES6 and the React framework, ensuring a modern, responsive, and intuitive user experience. STAR uses Terraform for Infrastructure as Code (IaC) to support automated deployment and configuration management. These architectural elements provide a powerful, flexible foundation for delivering robust, real-time insights into software quality across complex, multi-release environments.

3. STAR User Input Data

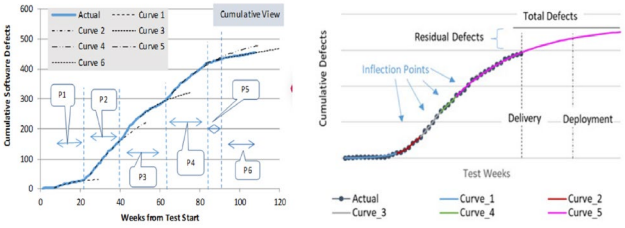

FIGURE 4 - STAR ARRIVAL CURVE - LOGIC

STAR provides a user-friendly, menu-driven interface to simplify the configuration and uploading of data for analysis. To begin, users navigate the release configuration page by selecting the "User Inputs" menu and clicking "Release Config." This page is organized into two main sections: one for entering release parameters and another for uploading defect data. In the release parameters section, users can enter key project milestone dates. These include the project start date, D1 (Delivery for Trial), and D2 (Delivery for Deployment). Accurate input of these dates is essential for aligning STAR's analytics with the project timeline.

Users should click the "Download Sample Defects" link to upload defect data. This action provides a downloadable template, shown in Figure 8, which outlines the required data format. Users must populate this template with their defect data, ensuring it conforms to the format and instructions provided at the bottom of the page. Once the defect data has been saved in CSV format, users can select the file using the "CHOOSE CSV FILE" button. After selecting the appropriate file, clicking the "UPLOAD DEFECTS" button initiates the upload process.

Upon successful upload, STAR automatically displays the uploaded data below the upload box for user verification.

This visual feedback lets users confirm that their data has been correctly formatted and processed. The same upload procedure applies to effort data, which is used for early defect prediction and further analytics within the STAR platform.

4. STAR Analytics

a. Defect Trend Analysis

This section presents innovative methods for predicting defect arrival, closure, and open curves. We also provide sample STAR outputs.

- Defect Arrival Curve

Traditional software reliability models often use a single-curve approach to represent defect arrivals. In contrast, STAR implements a multi-curve method based on multiple exponential models, each corresponding to a specific development phase. This is particularly useful when features are gradually introduced, resulting in distinct waves of defect arrivals. An algorithm automatically detects inflection points—where the trend shifts—and applies maximum likelihood estimation to determine model parameters. The result is a highly accurate piecewise exponential fit that outperforms single-curve models. Figure 4 illustrates the logic of piecewise exponential curves. Figure 3 shows a sample output of the STAR arrival curve, which appears near-perfect.

Theoretically, for each period *i*, the cumulative defects $m_i(t)$ are modeled as:

$$m_i(t) = a_i[1 - \exp\{-b_i(t - t_{i-1})\}] + m_{i-1}(t_{i-1}) \quad \forall\, t_{i-1} < t < t_i \qquad (4)$$

where a_i and b_i are parameters for total defects and defect rate, and t lies between two inflection points, t_{i-1} and t_i. This algorithm's flexibility accommodates various data shapes and sizes.

- Defect Closure Curve

Closure curve predictions are derived by shifting the predicted arrival curve to the right, based on actual closure data. Typically, the last two closure data points determine the optimal shift. Figure 3 illustrates the arrival and closure curves, which help demonstrate how closely the closure curve aligns with the arrival curve.

- Defect Open Curve

The open defect count—also known as backlog—is calculated as the difference between the arrival and closure curves:

$$\text{Open Defects} = \text{Arrival Defects} - \text{Closure Defects} \qquad (5)$$

This metric is crucial for determining readiness for delivery. Figure 3 shows the strong alignment between actual and predicted open defect curves.

b. Key Metrics for Software Quality Assurance

Predicted arrival, closure, and open curves enable the calculation of essential quality metrics:

- % Residual Defects

We standardize residual defects by the total number of defects, making this metric applicable to a diverse range of projects. Its definition is as follows:

$$\%\, Residual\ Defects = \frac{Total\ Defects - Defects\ Found}{Total\ Defects} \qquad (6)$$

Residual defects are expressed as a percentage of the total predicted defects: Acceptable (Green): ≤ 15%, Warning (Yellow): 15%–25%, At Risk (Red): > 25%.

- % Open Defects

The defects found normalize open defects as:

$$\%\, Open\ Defects = \frac{Defects\ Found - Defects\ Fixed}{Defects\ Found} \qquad (7)$$

Our recommended threshold values, grounded in practical experience, are as follows: Acceptable (Green): ≤ 5%, Warning (Yellow): 5%–10%, At Risk (Red): > 10%.

- Release-Readiness Assessment

Combining the residual and open defect metrics allows a visual readiness assessment. STAR calculates critical metrics such as %Residual Defects and %Open Defects to help evaluate whether a software release meets the desired quality thresholds. These metrics are illustrated in Figure 3, providing clear indicators of release preparedness at key milestones, such as D1 and D2. In the example, D2 (Deployment) meets quality standards, while D1 (Trial) does not.

c. Early Defect Prediction for Software Release Planning

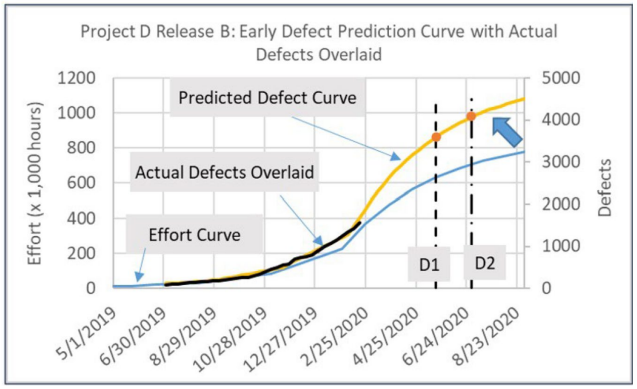

FIGURE 5 – EARLY DEFECT PREDICTION WITH ACTUAL DATA

Conventional SRGMs rely on actual defect data, limiting their utility in early planning. STAR addresses this by forecasting defects using early effort data, which is crucial for resource allocation and risk management in fast-paced environments.

- Input Data for Early Prediction

Effort data serves as a proxy for early defect prediction, as development effort reflects the complexity and content growth, and test effort mirrors test progress, correlating with defect detection.

- Preprocessing Effort Data

Effort data is transformed to enable prediction: (a) Normalize Development Effort (NDE) by test progress, (b) Compute Defect Density (DD), which is defects per effort hour, and (c) Calculate Target Defects (TD) at D1 and D2 by multiplying effort hours by DD.

- Transformation Algorithm

A transformation function shifts the normalized effort curve horizontally (α, β) and vertically (γ) to match the TD values. Equations (12) and (13) define the mapping from (x, y) coordinates to (x_{new}, y_{new}).

$$x_{new} = \alpha + \beta x \quad (12)$$
$$y_{new} = \gamma y \quad (13)$$

The optimization minimizes the difference between predicted and target defects. This three-variable nonlinear problem is solved numerically using nested iterations. Figure 5 shows high accuracy between early predictions and defect data. STAR automates the early defect prediction process and predicts potential defect patterns using development and test effort data even before defect data becomes available.

BIOGRAPHY

Dr. Kazuhira (Kazu) Okumoto, Ph.D. in Industrial Engineering and Operations Research from Syracuse University (1979), is a nationally recognized expert in software reliability with decades of experience in academia, industry, and federal R&D. After retiring from Bell Labs, he founded Sakura Software Solutions and developed cloud-based tools for software quality and system reliability. His vision is to transform how organizations assess reliability, enabling faster, safer, and informed decision-making throughout the software lifecycle.

Kazuhira Okumoto, PhD, CEO
Sakura Software Solutions, LLC
828 Heatherton Drive
Naperville, IL 60563 USA

Email: kokumoto@sakurasoftsolutions.com

REFERENCES

1. https://en.wikipedia.org/wiki/Digital_transformation
2. Aarti Phapte, "Global Software Quality Assurance Market Research Report Information by Solution," Aug. 2023. Accessed: Aug. 26, 2023. [Online]. Available: https://www.marketresearchfuture.com/reports/software-quality-assurance-market-8386
3. Z. Jelinski, P. B. Moranda, "Software reliability research," *Statistical Computer Performance Evaluation*, Feiger, W., editor, Academic Press, 1972. pp. 485-502, ISBN 9780122669507, https://doi.org/10.1016/B978-0-12-266950-7.50029-3. (http://www.sciencedirect.
4. J. D. Musa, "A theory of software engineering and its application," *IEEE Transactions on Software Engineering*, SE-1. 1975;3:312-327
5. L. Goel and K. Okumoto, 'Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures', *IEEE Transactions on Reliability*, vol. R-28, no. 3, 1979, doi: 10.1109/TR.1979.5220566.
6. H. Pham, *Software Reliability*, Berlin, Heidelberg: Springer-Verlag, 1999.
7. D. Sudharson and D. Prabha, "A novel machine learning approach for software reliability growth modelling with Pareto distribution function," *Soft Comput*, vol. 23, no. 18, pp. 8379–8387, 2019, doi: 10.1007/s00500-019-04047-7.
8. S. Yamada, M. Ohba, and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability*, vol. R-32, no. 5, pp. 475–484, 1983, doi: 10.1109/TR.1983.5221735.
9. B. Littlewood and J.L. Verrall, 'A Bayesian reliability growth model for computer software', Journal of the Royal Statistical Society, Series C (Applied Statistics), vol. 22, no. 3, pp. 332 – 346, 1973.
10. J.D. Musa and K. Okumoto, 'A Logarithmic Poisson Execution Time Model for Software Reliability Measurements', *Proceedings Seventh International Conference on Software Engineering*, Orlando, pp. 230 – 238, 1984.
11. D. R. Jeske, X. Zhang, and L. Pham, "Adjusting software failure rates that are estimated from test data," *IEEE Transactions on Reliability*, vol. 54, no. 1, 2005, doi: 10.1109/TR.2004.842531.
12. H. Pham, "A logistic fault-dependent detection software reliability model," *Journal of Universal Computer Science*, vol. 24, no. 12, 2018.
13. J. Sorrentino, P. Silva, G. Baye, G. Kul, and L. Fiondella, "Covariate Software Vulnerability Discovery Model to Support Cybersecurity Test & Evaluation (Practical Experience Report)," *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 157–168. doi: 10.1109/ISSRE55969.2022.00025.
14. J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, USA: McGraw-Hill, Inc., 1987.
15. K. Okumoto, "Digital Engineering-driven Software Quality Assurance-as-a-Service," *ISSAT International Conference on Reliability and Quality in Design*, 2023.
16. K. Okumoto, "Early Software Defect Prediction: Right-Shifting Software Effort Data into a Defect Curve," *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2022, pp. 43–48. doi: 10.1109/ISSREW55968.2022.00037.
17. K. Okumoto, A. Asthana, and R. Mijumbi, "BRACE: Cloud-based software reliability assurance," *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* 2017. doi:

10.1109/ISSREW.2017.48.
18. R. Mijumbi, K. Okumoto, A. Asthana, and J. Meekel, “Recent Advances in Software Reliability Assurance,” *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018, pp. 77–82. doi: 10.1109/ISSREW.2018.00-27.
19. R. Mijumbi, K. Okumoto, and A. Asthana, “Software Reliability Assurance in Practice,” *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2019. doi: 10.1002/047134608x.w6952.pub2.