System Reliability Intelligence Modeling and Tool

Kazuhira Okumoto[1*] and Hoang Pham[2]

[1*]Sakura Software Solutions LLC, Naperville, 60563, Naperville, USA.
[2]Department of Industrial and Systems Engineering, Rutgers University, Piscataway, 08854, New Jersey, USA.

*Corresponding author(s). E-mail(s): kokumoto@sakurasoftsolutions.com;
Contributing authors: hopham@soe.rutgers.edu;

Abstract

In an era where AI, automation, and robotics are pervasive, system failures can lead to severe repercussions, making the reliability of software and hardware crucial. Current reliability approaches fall short in providing comprehensive models for the operational phase of software. This chapter introduces a unified reliability model that integrates software and hardware components, emphasizing the operational phase of the software lifecycle. We also present FUSION, a cloud-based digital platform that leverages our proposed model. FUSION offers a cohesive user interface for running various system reliability configuration scenarios, providing metrics that inform design decisions. Beyond issue detection, FUSION facilitates a proactive approach to identifying and mitigating potential issues, thereby enhancing overall system reliability and customer satisfaction. Our work bridges the gap in existing reliability models and delivers a robust solution for the dynamic technological landscape.

Keywords: Reliability modeling, Operational phase, Software and hardware failures, Cloud

1 Introduction

Ensuring software and hardware reliability is crucial in a world increasingly reliant on AI, automation, and robotics [1]. The focus on reliability is essential because even minor defects can lead to significant disruptions. Numerous models [2–11] exist for predicting software defects during coding and testing phases. These and other models are

Submitted for a book chapter in the Springer Handbook of Reliability, ed. H. Pham, 2025

summarized in Appendix A. However, a significant gap still remains in the operational phase—when the software is deployed to customers. This phase represents real-world use of the software, making it particularly critical. Defects that arise during this phase can lead to system downtime and negatively impact users. Moreover, because software is typically deployed on hardware, failures can still occur if the underlying hardware fails, even when the software is reliable. Therefore, it is necessary to analyze system reliability, considering both software and hardware components. On its own, hardware reliability during the operational phase is well-studied [12–19]. However, studies that consider an integrated system involving both software and hardware are limited. The interaction between software and hardware adds complexity to reliability assessments, necessitating comprehensive models that can account for potential failures in both domains. Integrating these aspects is essential for developing more robust systems that can withstand the challenges of real-world applications.

In this chapter, we develop and implement cutting-edge algorithms in FUSION's innovative cloud-based tool, which plays a pivotal role by combining software and hardware models to analyze defects and failures during the operational phase. Our research demonstrates that software failure rates can be modeled with a constant rate during this phase, aligning with established hardware failure models to create a unified reliability model. This unified model not only addresses a significant gap in the field but also has the potential to revolutionize how we approach software and hardware reliability. In the rapidly evolving technological landscape, this research is essential, given the interconnected failures of software and hardware. This, in turn, enhances system performance and ultimately improves customer satisfaction.

Our rigorous research process begins by correlating software reliability growth models with the hardware bathtub curve in Fig. 1. By focusing on a constant defect growth rate during the operational phase, we align it with the flat phase of the hardware curve. Using observed software failure rates during customer operations helps establish this correlation. We propose integrated models, defining system reliability

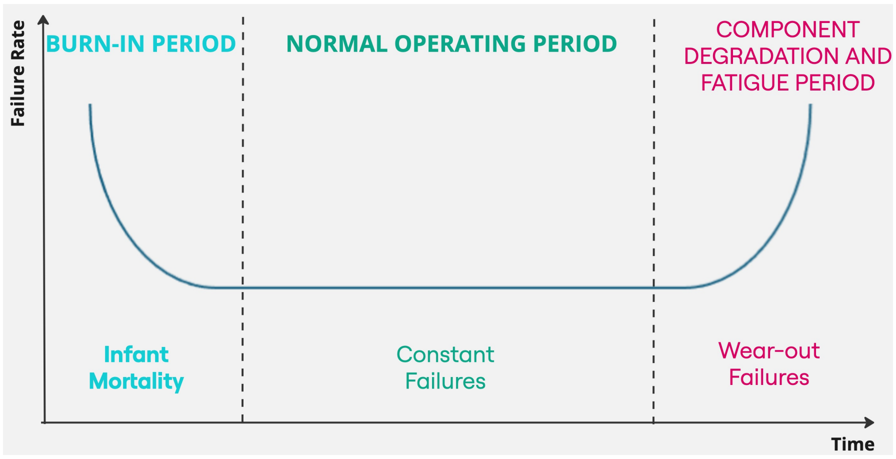


Fig. 1: Hardware Bathtub Curve

through reliability block diagrams (RBDs) [17, 18, 20] and operational reliability and availability metrics.

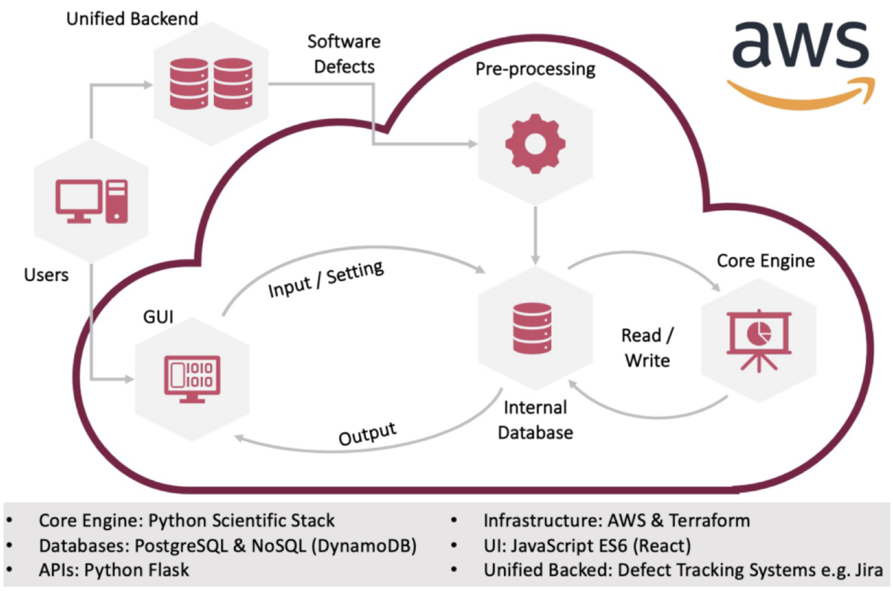
FUSION, built with advanced analytics and visualization techniques, forecasts software failure rates by incorporating defect and failure data. A key feature is the graphical editor, which allows users to create system RBDs, including both software and hardware components. This tool accelerates system reliability evaluation, identifies bottlenecks, aids in practical improvements, and enhances overall system performance.

Our research is driven by a clear goal: revolutionizing software quality and reliability. We aim to align software defect modeling with hardware failures during customer operations. The comprehensive digital tool we propose, FUSION, will empower practitioners to improve reliability, leading to greater customer satisfaction and profitability. This is not just a research project; it's a mission to reshape the way we think about software and hardware reliability, and we invite you to be a part of it.

The main contributions of this chapter are three-fold: (1) a combined software and hardware reliability model, (2) algorithms for automated reliability analysis, and (3) a digital tool for system-level reliability assessments.

2 Fusion System Architecture

FUSION streamlines the entire workflow, from data extraction and pre-processing to core analytics and post-processing. Figure 2 presents the high-level architecture of FUSION, which is built on the AWS platform. FUSION employs APIs (such as Flask) to gather data from various defect-tracking tools and consolidate the data into two databases—PostgreSQL and NoSQL (DynamoDB)—to optimize performance. Prior to



- Core Engine: Python Scientific Stack
- Databases: PostgreSQL & NoSQL (DynamoDB)
- APIs: Python Flask
- Infrastructure: AWS & Terraform
- UI: JavaScript ES6 (React)
- Unified Backed: Defect Tracking Systems e.g. Jira

Fig. 2: FUSION System Architecture

storage, the data undergoes pre-processing, which involves aggregating it into weekly or specified time frames and preparing the necessary input for the core analytics engine that generates predictions. A critical part of pre-processing is ensuring data consistency both across different projects and within the same project over multiple releases. For example, pre-processing may include:

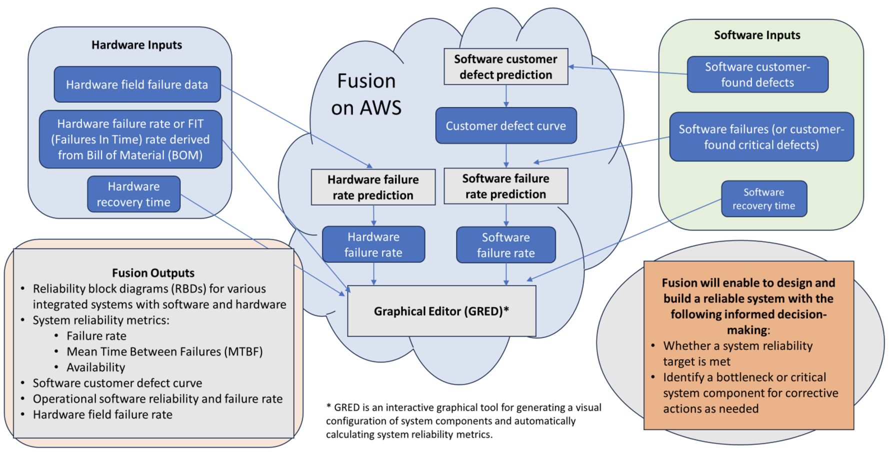
- Standardizing database field and value mappings, as different projects or databases might use varying field names for the same attributes (e.g., priority vs. severity).
- Mapping certain field values to specific categories, such as assigning a defect to a geographic region instead of an organizational unit.
- Checking defect properties to detect duplicates or defects from other releases aids in quality control and project management. The core analytics engine is developed using the Python Scientific Stack, while the user interface is built using JavaScript ES6 (React). Terraform AWS manages the system's infrastructure as code (IaC).

3 Fusion Overview

This section will provide an overview of FUSION, a comprehensive cloud-based tool that enhances system reliability (see Figure 3). FUSION comprises two critical components: software failure rate prediction model and an interactive graphical editor, GRED, for constructing various reliability block diagrams (RBDs).

3.1 Software customer-found defect prediction

The operational phase of software usage can be divided into distinct periods: early customer testing, customer testing, post-deployment, and customer migration to the next release. Each of these periods exhibits varying rates of defect identification (see Figure 4). To capture this variability, we aim to construct a defect trend model that spans the entire operational phase, represented as a sequence of empirically derived straight


Hardware Inputs
Hardware field failure data
Hardware failure rate or FIT (Failures In Time) rate derived from Bill of Material (BOM)
Hardware recovery time
Fusion on AWS
Software customer defect prediction
Customer defect curve
Hardware failure rate prediction
Software failure rate prediction
Hardware failure rate
Software failure rate
Graphical Editor (GRED)*
Software Inputs
Software customer-found defects
Software failures (or customer-found critical defects)
Software recovery time
Fusion Outputs
Reliability block diagrams (RBDs) for various integrated systems with software and hardware
System reliability metrics:
Failure rate
Mean Time Between Failures (MTBF)
Availability
Software customer defect curve
Operational software reliability and failure rate
Hardware field failure rate
Fusion will enable to design and build a reliable system with the following informed decision-making:
Whether a system reliability target is met
Identify a bottleneck or critical system component for corrective actions as needed
* GRED is an interactive graphical tool for generating a visual configuration of system components and automatically calculating system reliability metrics.
Fig. 3: An overview of FUSION

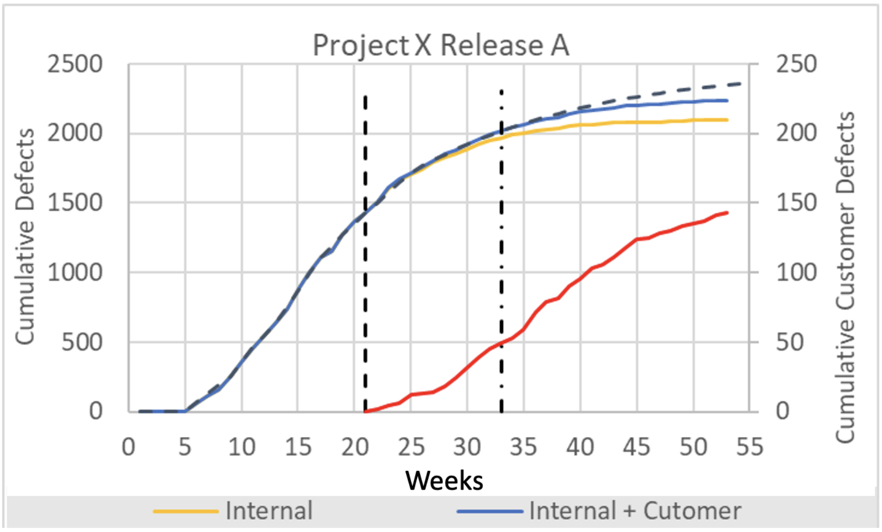


Fig. 4: Software defect curves: Internal vs. customer

lines. The primary technical challenge is developing an algorithm that autonomously identifies the transition points (inflection points) between each straight line. This algorithm must adapt to the distinct patterns within the data and function seamlessly in real-time as new defect data becomes available.

We have developed an algorithm that automatically detects inflection points and provides a segmented linear representation of software defects throughout the operational phase. A segmented linear equation is expressed in mathematical form as:

$$m(x) = m(x_{j-1}) + \theta_j \big(x - x_{j-1}\big) \qquad \forall x_{(j-1)} \leq x \leq x_j \tag{1}$$

Figure 5 showcases these sequential lines and visually depicts the algorithm's result. For enhanced comprehension, it also offers a weekly breakdown of the trend. Details of the algorithm are provided in Appendix B.

This algorithm's applicability significantly extends beyond software, including hardware failure data. Additionally, we will compare the observed hardware failure rate against the rate projected from the bill of materials (BOM) to provide comprehensive insights. This approach helps accurately model defect trends and ensures timely and effective defect management throughout the software lifecycle.

3.2 Software Failure Rate Prediction

FUSION introduces innovative methods for predicting software defects and failures, demonstrating that the software defect and failure rates remain constant during the operational period. This capability is crucial as it aligns software reliability modeling with established hardware reliability models, providing a unified approach to system reliability. The analytics for predicting software defects are robust enough to be applied to hardware field failure data, offering a versatile solution for reliability analysis across

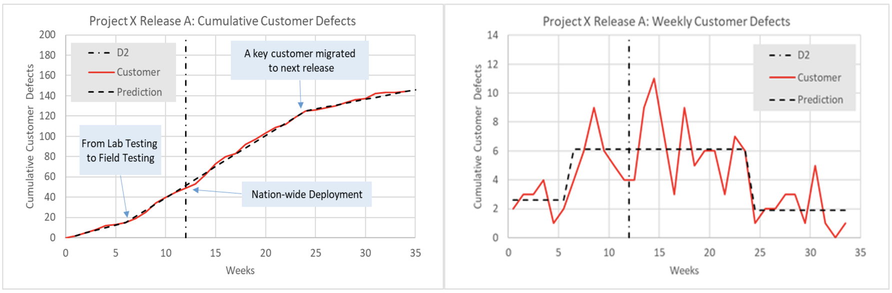


Fig. 5: Customer-found defect curves with a prediction: Cumulative vs. Weekly

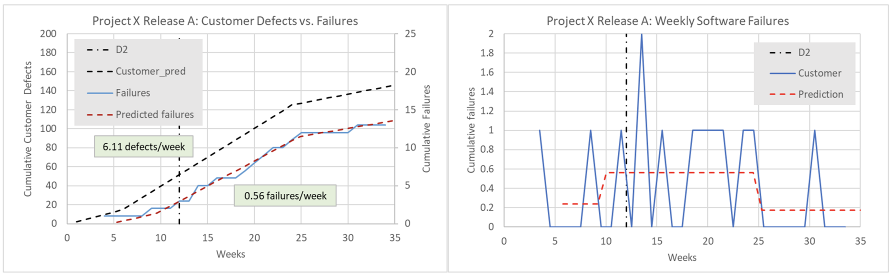


Fig. 6: Predicted software failure curves: Cumulative vs. weekly

different system components. We have developed an innovative algorithm to predict software failure rates during the operational phase, using a transformation function that maps the defect prediction curve onto the software failure curve. Detailed information about the algorithm is provided in Appendix C. Figure 6) illustrates the predicted software failure curve alongside the actual data, presented in both cumulative and weekly views. It demonstrates that the software failure rate remains constant after deployment. This method has been applied to data from several projects, and all cases consistently confirm these findings. One advantage of this method is its effectiveness even with a very small number of failures, such as fewer than 10.

3.3 Interactive Graphical Editor (GRED)

GRED is an interactive graphical editor embedded within FUSION that allows users to construct detailed RBDs for software and hardware components. GRED features a built-in function for automatically calculating reliability metrics for combined software and hardware systems and individual components. This functionality simplifies the complex reliability assessment process, enabling users to quickly design and build more reliable systems.

3.4 System Design and Building

FUSION empowers users to design and build reliable systems by integrating advanced analytics and interactive modeling tools. The software failure rate prediction component provides accurate defect and failure rate forecasts, crucial for preemptive maintenance and system optimization. GRED's intuitive interface and automatic calculation features facilitate the creation of comprehensive RBDs, ensuring that users can effectively evaluate and improve system reliability.

3.5 Summary and Future Discussions

Figure 3 summarizes FUSION's input and output data, analytics, and GRED. This figure illustrates how FUSION integrates data processing and visualization to provide a holistic view of system reliability.

In section 4, we will delve deeper into the specifics of these components. Specifically, we discuss GRED, highlighting its features, user interface, and the automatic calculation of reliability metrics.

By combining advanced predictive analytics with an intuitive graphical interface, FUSION offers a powerful tool for system reliability engineers. It enables them to anticipate and mitigate potential failures effectively. This integrated approach enhances the reliability of individual software and hardware components and ensures the overall robustness of the entire system.

4 Interactive Graphical Editor (GRED) for Reliability Block Diagrams (RBDs)

Reliability engineers often use reliability block diagrams (RBDs) to model and analyze complex systems. RBDs visually represent the system's configuration and components, simplifying system reliability calculation through series and parallel equations. This section outlines the initial design of a digital application, GRED, that integrates software and hardware components, enabling the interactive creation of system RBDs. GRED will provide users with several preconfigured RBD templates, including a single-unit system, a two-unit active-standby system, and a two-unit active-active load-sharing system, as illustrated in Fig. 7. By clicking a template, it will ask for input data of failure rate and recovery time for hardware and software. Once the input data is completed and the submit box is clicked, it will calculate and display the outputs of reliability metrics. We will discuss the process of creating these diagrams, the required inputs, and how to derive reliability metrics for each configuration. This tool aims to streamline the design and analysis process, making it easier for users to evaluate and enhance system reliability.

4.1 Reliability metrics

The **failure rate** measures how often a system, component, or device fails over a specified period. It is typically denoted by the symbol λ and expressed as the number of failures per unit of time (e.g., failures per hour, per day, or per year). The failure rate is a key parameter in reliability engineering and plays a crucial role in assessing and predicting system **reliability** and **availability**. Understanding and calculating the failure rate is fundamental for designing, maintaining, and improving the reliability of systems and components, providing engineers with vital information for their work.

A low failure rate indicates that the system or component is reliable and expected to fail infrequently. A high failure rate means the system or component is less reliable and is expected to fail more frequently. There are two types of failure rates. The constant failure rate, often seen in electronics and mechanical systems, is assumed to be constant during the useful life period (the middle part of the bathtub curve in Fig. 1). On the other hand, the time-varying failure rate is when failure rates may change over time, especially when systems are subject to wear-out (increasing failure rate) or early failures (decreasing failure rate).

The failure rate is used in calculating the reliability function ($R(t)$), which is the probability that a system operates without failure over a specified period of time t. For example, the reliability function $R(t)$ for a system with a constant failure rate follows the exponential distribution:

$$R(t) = e^{-\lambda t} \tag{2}$$

Mean Time to Failure (MTTF) and Mean Time Between Failures (MTBF) measure systems or components' reliability. However, it's crucial to understand that they differ in their application and interpretation. MTTF represents the average time a non-repairable system or component operates before it fails. It is used for systems or components that are non-repairable, meaning that when they fail, they are replaced rather than repaired. MTTF indicates the expected operational lifespan of a component before it fails. A higher MTTF value implies that the component is more reliable and can be expected to last longer. It is calculated as follows:

$$MTTF = \int_0^\infty R(t)dt \tag{3}$$

For a constant failure rate, the MTTF is given by:

$$MTTF = \frac{1}{\lambda} \tag{4}$$

MTTF is used for components like light bulbs, batteries, and other products that are typically discarded or replaced after failure.

It is calculated as:

$$MTTF = \frac{\text{Total Operational Time}}{\text{Number of Failures}} \tag{5}$$

MTBF represents the average time between successive failures of a system that can be repaired and returned to service. It is used for repairable systems like servers, machinery, and other systems routinely repaired and returned to service. MTBF measures when a system starts functioning after a repair and lasts until the subsequent failure occurs. MTTR (Mean Time to Repair) is the average time to repair and restore the system to operational status after a failure. MTBF is the sum of MTTF and MTTR for a repairable system:

$$MTBF = MTTF + MTTR \tag{6}$$

A higher MTBF value indicates that the system is more reliable because it runs longer on average between failures. It helps plan maintenance schedules and estimate system uptime.

Availability measures a system's ability to perform its intended function when needed. It is expressed as the proportion of time that a system is operational and ready for use compared to when it is expected to be available. Availability considers the system's uptime and downtime, accounting for maintenance, repairs, or any other factors that may cause the system to be non-operational. The basic formula for availability (A) is:

$$A = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}} \tag{7}$$

or, equivalently:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{8}$$

High Availability (close to 1) indicates that the system is usually operational with minimal downtime. High availability systems are often designed to ensure that the probability of being down is very low. Low availability (closer to 0) indicates that the system is down frequently or has long repair times, reducing operational time.

Availability is applied to the following areas:

- System Design: Used to evaluate and design systems that need to meet specific operational availability requirements (e.g., telecommunications, military, and healthcare systems).
- Maintenance Planning: Helps optimize maintenance schedules and resources to maximize the operational time of equipment.
- Service Level Agreements (SLAs): Often expressed in availability terms (e.g., "99.9% or three 9s uptime") to set expectations for system performance and reliability.

Availability is a critical metric in industries where continuous operation is essential and minimizing downtime is a priority.

4.2 Templates for reliability block diagram

The following three configurations are considered for the Phase I development. See Figure 7. Other configurations will be included in the Phase II development.

In table 1 we show details of how FUSION metrics: failure rate (λ), Mean Time To Failure (MTTF), Reliability (% number of failures in one year), Mean Time between Failures (MTBF), Mean Time to Repair (MTTR), and Availability are determined for software, hardware, and system (HW + SW).

4.2.1 A single-unit system with software and hardware

Fig. 7 highlights GRED's basic templates. Selecting the appropriate template displays a single-unit RBD with software and hardware components on the screen. GRED then

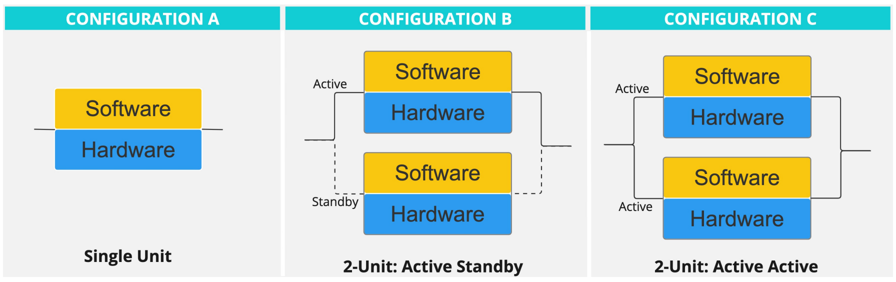


Fig. 7: Set of Basic Templates/Configurations for the GRED

Table 1: FUSION Output Metrics Calculation Formulas for Configuration A

Metric	Formulas		
	Hardware	Software	System
Failure Rate	λ_{hw}	λ_{sw}	$\lambda_{hw} + \lambda_{sw}$
MTTF	$\frac{1}{\lambda_{hw}}$	$\frac{1}{\lambda_{sw}}$	$\frac{1}{\lambda_{sys}}$
Reliability	$e^{-\lambda_{hw}t}$	$e^{-\lambda_{sw}t}$	$e^{-\lambda_{sys}t}$
MTBF	$MTTF_{hw} + MTTR_{hw}$	$MTTF_{sw} + MTTR_{sw}$	$MTTF_{sys} + MTTR_{sys}$
MTTR	$\frac{1}{\mu_{hw}}$	$\frac{1}{\mu_{sw}}$	$\frac{1}{\mu_{sys}} = \left(\frac{\lambda_{hw}}{\mu_{hw}} + \frac{\lambda_{sw}}{\mu_{sw}}\right)/\lambda_{sys}$
Availability	$\frac{MTTF_{hw}}{MTBF_{hw}}$	$\frac{MTTF_{sw}}{MTBF_{sw}}$	$\frac{MTTF_{sys}}{MTBF_{sys}}$

[a] μ is the repair rate and is measured in **repairs per unit time**

prompts for input data, including the software and hardware failure rates and recovery times. Software and hardware failure rates are constant during the operation phase, denoted by λ_{sw} and λ_{hw}, respectively. We can treat the software and hardware as components in a series system for a single-unit configuration.

In a series system, all components must function correctly for the entire system to operate. In general, we can express the reliability of a series system as a product of the reliability of individual components. Mathematically, the reliability for a series system with n component with the reliability of component i, denoted by $R_i(t)$, can be expressed as:

$$R_{sys}(t) = \prod_{i=1}^{n} R_i(t) \tag{9}$$

Table 2: User Inputs for Configurations A, B and C

Input	Notation	Hardware	Software
Failure rate (failures/year)	λ	0.15	0.6
Manual recovery time (minutes)*	$\frac{1}{\mu}$	240	120

* Includes time for diagnostics, repair or replacement, hardware reboot for hardware recovery; troubleshooting, quick software fixes, software reboot for software recovery

Table 3: FUSION Outputs for Configuration A

Metric	Hardware	Software	System (SW + HW)
Failure rate	0.15	0.60	0.75
MTTF	6.6667	1.6667	1.3333
Reliability	86.1%	54.9%	47.2%
MTBF	6.66671	1.6669	1.3336
MTTR	240	120	144
Availability	99.993%	99.986%	99.979%

For exponential distribution with a failure rate λ_i

$$R_i(t) = e^{-\lambda_i t} \tag{10}$$

We have

$$R_{sys}(t) = e^{-\sum_{i=1}^{n} \lambda_i t} \tag{11}$$

In other words, the system failure rate is a sum of the failure rates of individual components.

$$\lambda_{sys} = \sum_{i=1}^{n} \lambda_i \tag{12}$$

The system failure rate, denoted by λ_{sys}, is the sum of the failure rates of the software and hardware, expressed as (13)

$$\lambda_{sys} = \lambda_{sw} + \lambda_{hw} \tag{13}$$

Mean Time to Failure (MTTF) applies to non-repairable items and indicates the expected operational lifespan before a product fails. For a constant failure rate, MTTF can be calculated as the inverse of the failure rate, λ_{sys}:

$$MTTF_{sys} = \frac{1}{\lambda_{sys}} \tag{14}$$

Equation (14) can be applied to both software and hardware. System reliability measures the probability that a complex system, composed of multiple components, will function without failure over a specified period. Different ways to calculate system reliability depend on the configuration and components' characteristics. We assume that time to failure follows an exponential distribution with a constant failure rate

λ_{sys}. The reliability of a system with the failure rate of λ_{sys} during t year can be calculated as:

$$R_{sys}(t) = e^{-\lambda_{sys}t} \tag{15}$$

In a series configuration with repairable units, the system fails when any one of the three units fails. Thus, MTTR accounts for the weighted contributions of the repair times of individual components, assuming each has a different repair rate μ_i, failures repaired per unit time. $MTTR_i$, the mean time to repair unit i, is the reciprocal of the repair rate: $MTTR_i = 1/\mu_i$. Since the system fails when any one of the units fails, the MTTR for the system is the expected repair time, considering the probability of each unit failing, λ_i/λ_{sys}. The MTTR for the system is the weighted sum of the individual repair times, weighted by the probability of each unit failing:

$$MTTR_{sys} = \sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda_{i_{sys}}} \times \frac{1}{\mu_i} \right) \tag{16}$$

The system's MTBF can be calculated using (6) by substituting (14) and (16).

We introduce the expected repair or recovery time to calculate the system availability: $1/\mu_{sw}$ for software and $1/\mu_{hw}$ for hardware, respectively. The average system recovery time, $MTTR_{sys}$, can be found as the weighted sum of the failure rates:

$$MTTR_{sys} = \frac{1}{\mu_{sys}} = \frac{\frac{\lambda_{sw}}{\mu_{sw}} + \frac{\lambda_{hw}}{\mu_{hw}}}{\lambda_{sys}} \tag{17}$$

In practice, we measure the downtime in minutes. The annual downtime is adjusted accordingly. The system availability is calculated using (8) by substituting (14) and (17):

$$A_{sys} = \frac{\frac{1}{\lambda_{sys}}}{\frac{1}{\lambda_{sys}} + \frac{1}{\mu_{sys}}} \tag{18}$$

The above system reliability metrics can be derived separately from (14) through (18) for software and hardware. Once the user input data is completed (see Table 1), FUSION will automatically compute the reliability metrics and display the table.

Table 1 summarizes the formulas for the reliability and availability metrics for Configuration A. Table 3 shows the Fusion output of the metrics.

4.2.2 A two-unit active-standby (cold) system

To calculate the MTTF for a two-unit active-standby system, we consider each unit's failure rate and the fact that the system switches to the standby unit when the active unit fails. The following assumptions and definitions are made:

- Each unit has a failure rate of λ (failures per unit time).
- The units are identical and independent.
- The system is configured so that one unit (Unit A) is active while the other (Unit B) is in standby mode, and the standby unit becomes active immediately upon the failure of the first.

Table 4: FUSION Output Metrics Calculation Formulas for Configuration B

Metric	Formulas		
	Hardware	Software	System
Failure Rate	$2 \times \lambda_{hw}$	$2 \times \lambda_{sw}$	$2 \times (\lambda_{hw} + \lambda_{sw})$
MTTF	$\frac{1}{2\lambda_{hw}}$	$\frac{1}{2\lambda_{sw}}$	$\frac{1}{2\lambda_{sys}}$
Reliability	$e^{-2\lambda_{hw}t}$	$e^{-2\lambda_{sw}t}$	$e^{-2\lambda_{sys}t}$
MTBF	$\frac{MTTF_{hw}}{A_{hw}}$	$\frac{MTTF_{sw}}{A_{sw}}$	$\frac{MTTF_{sys}}{A_{sys}}$
MTTR	$\frac{1}{\mu_{hw}}$	$\frac{1}{\mu_{sw}}$	$\frac{1}{\mu_{sys}} = \left(\frac{\lambda_{hw}}{\mu_{hw}} + \frac{\lambda_{sw}}{\mu_{sw}}\right)/\lambda_{sys}$
Availability	$\frac{1+\frac{\lambda_{hw}}{\mu_{hw}}}{1+\frac{2\lambda_{hw}}{\mu_{hw}}+2\left(\frac{\lambda_{hw}}{\mu_{hw}}\right)^2}$	$\frac{1+\frac{\lambda_{sw}}{\mu_{sw}}}{1+\frac{2\lambda_{sw}}{\mu_{sw}}+2\left(\frac{\lambda_{sw}}{\mu_{sw}}\right)^2}$	$\frac{1+\frac{\lambda_{sys}}{\mu_{sys}}}{1+\frac{2\lambda_{sys}}{\mu_{sys}}+2\left(\frac{\lambda_{sys}}{\mu_{sys}}\right)^2}$

[a] μ is the repair rate and is measured in **repairs per unit time**

Table 5: FUSION Outputs for Configuration B

Metric	Hardware	Software	System (SW + HW)
Failure rate	0.15	0.60	0.75
MTTF	13.333	3.333	2.6667
Reliability	99.0%	87.8%	82.7%
MTBF	13.334	3.334	2.6669
MTTR	240	120	144
Availability	99.997%	99.993%	99.990%

- We assume that the switch from the standby to active unit happens without delay and that the switching mechanism is 100% reliable.
- The standby unit has a negligible failure rate while in standby mode, a cold standby units (ideal standby assumption).

Each unit's MTTF is $1/\lambda$ since MTTF is the reciprocal of the failure rate from (3). To calculate the system MTTF, we consider two stages: 1) The time until the first unit fails (the active unit) and 2) The time until the second unit (standby) fails after taking over. Since both units have the same MTTF and the above stages are mutually exclusive, the system MTFF is the sum of the MTTFs of both units:

$$MTTF_{sys} = \frac{1}{\lambda} + \frac{1}{\lambda} = \frac{2}{\lambda} \tag{19}$$

This result indicates that the MTTF of the two-unit active-standby system is double that of a single unit with the same failure rate, reflecting the added reliability of having a backup unit ready to take over. The system reliability function of this configuration is derived as follows. The system reliability, $R_{sys}(t)$, is the probability that either 1) Unit A survives for the entire duration t, or 2) Unit A fails at some time t_1, and then Unit B takes over and survives for the remaining time.

Case 1: Unit A Survives Entire Duration t: The probability that Unit A does not fail by time t is

$$R_A(t) = e^{-\lambda t} \tag{20}$$

Case 2: Unit A Fails, and Unit B Takes Over: By creating the joint probability that Unit A fails at time t_1 and Unit B operates successfully for the remaining time $(t - t_1)$ and Integrating over all possible failure times for Unit A with t_1 from 0 to t, we derive the probability of Case 2. We can then derive the system reliability as the sum of the probabilities of the two cases:

$$R_{sys}(t) = e^{-\lambda t}(1 + \lambda t) \tag{21}$$

This function shows that the reliability of a two-unit active-standby system is higher than that of a single-unit system because the standby unit provides redundancy, reducing the likelihood of system failure over time. Substituting (21) into (3), MTTF for this configuration is derived as the same as (19).

To calculate the availability of a two-unit active-standby system, we need to consider its MTBF and MTTR. We make the following additional assumptions.

- A repair rate is μ (repairs per unit time). MTTR for each unit is:

$$MTTR = \frac{1}{\mu} \tag{22}$$

- The system only becomes unavailable when the second unit fails before the first unit can be repaired.

Substituting (19) and (22) into (7):

$$\text{Availability} = \frac{\frac{2}{\lambda}}{\frac{2}{\lambda} + \frac{1}{\mu}} \tag{23}$$

where $\frac{2}{\lambda} + \frac{1}{\mu}$ is MTBF. Simplifying the expression:

$$\text{Availability} = \frac{2\mu}{2\mu + \lambda} \tag{24}$$

Reliability and availability metrics can be calculated for software, hardware, and systems using respective λ and μ. Tables 4 and 5 illustrate the formulas and outputs for the same input data as in Table 2. There has been a significant improvement in reliability and availability over a single-unit configuration.

4.2.3 A two-unit active-active load-sharing system

In this configuration, both units are active simultaneously and share the system's load equally or in a balanced manner. Both units work together to process the workload,

Table 6: FUSION Output Metrics Calculation Formulas for Configuration C

Metric	Formulas		
	Hardware	Software	System
Failure Rate	λ_{hw}	λ_{sw}	$\lambda_{sys} = \lambda_{hw} + \lambda_{sw}$
MTTF	$\frac{2}{\lambda_{hw}}$	$\frac{2}{\lambda_{sw}}$	$\frac{2}{\lambda_{sys}}$
Reliability	$(1 + \lambda_{hw} t)e^{-\lambda_{hw} t}$	$(1 + \lambda_{sw} t)e^{-\lambda_{sw} t}$	$(1 + \lambda_{sys} t)e^{-\lambda_{sys} t}$
MTBF	$MTTF_{hw} + MTTR_{hw}$	$MTTF_{sw} + MTTR_{sw}$	$MTTF_{sys} + MTTR_{sys}$
MTTR	$\frac{1}{\mu_{hw}}$	$\frac{1}{\mu_{sw}}$	$\frac{1}{\mu_{sys}} = \left(\frac{\lambda_{hw}}{\mu_{hw}} + \frac{\lambda_{sw}}{\mu_{sw}}\right)/\lambda_{sys}$
Availability	$\frac{MTTF_{hw}}{MTBF_{hw}}$	$\frac{MTTF_{sw}}{MTBF_{sw}}$	$\frac{MTTF_{sys}}{MTBF_{sys}}$

[a] μ is the repair rate and is measured in **repairs per unit time**

Table 7: FUSION Outputs for Configuration C

Metric	Hardware	Software	System (SW + HW)
Failure rate	0.30	1.20	1.50
MTTF	3.3333	0.8333	0.6667
Reliability	74.1%	30.1%	22.3%
MTBF	3.3336	0.8334	0.6668
MTTR	240	120	144
Availability	99.993%	99.986%	99.979%

distributing the tasks to balance the load. For simplicity, we assume that if one unit fails, the workload will be lost until it is recovered. If the load distribution is 50-50, the system will lose 50% of the total load. It is called a partial failure. Essentially, two units are operating independently with their assigned workload. Adding another unit enhances the system's performance.

The other option is for the other unit to continue operating, but it will need to take over the entire load, potentially increasing its workload. This setup is designed for high availability and performance, as it maximizes resource utilization by leveraging both units simultaneously. Note that a two-unit active-active system is intended for load distribution and efficiency, while a two-unit parallel system focuses on redundancy and reliability. We will consider this option in future work.

To calculate the availability of a two-unit active-active system, we must consider that the system continues to function at a 50% capacity even if one unit fails. We have a few key assumptions:

- Each unit has an individual failure rate λ (failures per unit time) and a repair rate μ (repairs per unit time). $MTTR = 1/\mu$.
- The failure rates of the units are independent, and repairs occur immediately when a unit fails.

The system can be in three possible states: both units functioning (State 1), one unit functioning (State 2), or System down (State 3).

1. Both Units Functioning (State 1): The failure rate for this state (where either unit can fail) is 2λ. This is the transition rate from State 1 to State 2.
2. One Unit Functioning (State 2): If one unit fails, the system still functions at a reduced 50% capacity. The system remains in this state until either: (a) The failed unit is repaired, returning the system to State 1. The transition rate is μ. (b) The remaining operational unit fails, causing total system failure, State 3. The transition rate from State 2 to State 3 is λ since only one unit operates.
3. System Down (State 3): If both units fail, the system is unavailable. The repair process continues until one failed unit is repaired, at which point the system transitions back to State 2 at a rate of μ.

To calculate the steady-state probabilities, we define P_1, P_2 and P_3 as the probability of the system being in State 1, State 2, and State 3, respectively. Transition rate equations at State 1 and State 3 are derived as:

$$2\lambda P_1 = \mu P_2 \tag{25}$$
$$\lambda P_2 = \mu P_3 \tag{26}$$

The sum of all probabilities must equal 1:

$$P_1 + P_2 + P_3 = 1 \tag{27}$$

Solving the equations (25), (26), and (27), we have the steady-state probabilities:

$$P_1 = \frac{1}{1 + \frac{2\lambda}{\mu} + \frac{2\lambda^2}{\mu^2}} \tag{28}$$

$$P_2 = \frac{2\lambda}{\mu} P_1 \tag{29}$$

$$P_3 = \frac{\lambda}{\mu} P_2 = \frac{\lambda}{\mu} \times \frac{2\lambda}{\mu} P_1 = \frac{2\lambda^2}{\mu^2} P_1 \tag{30}$$

The system is available in either State 1 (both units functioning) or 50% of State 2 (one unit functioning). Therefore:

$$\text{Availability} = P_1 + 0.5 P_2 \tag{31}$$

Substituting (28) and (29) into (31), we have the final availability:

$$\text{Availability} = \frac{1 + \frac{\lambda}{\mu}}{1 + \frac{2\lambda}{\mu} + \frac{2\lambda^2}{\mu^2}} \tag{32}$$

To calculate the MTTF for a 2-unit active-active load-sharing system, we consider that both units operate simultaneously and share the load. When one unit fails, the system is regarded as a partial failure. Since each unit's failure rate is λ, the failure rate of the two units is 2λ. MTTF is a reciprocal of the failure rate:

$$\text{MTTF} = \frac{1}{2\lambda} \tag{33}$$

Since $A = MTTF/MTBF$, MTBF can be calculated as: $MTBF = MTTF/A$. Reliability and availability metrics can be calculated for software, hardware, and systems using respective λ and μ. Tables 6 and 7 illustrates the formulas and outputs for the same input data as in Table 2. It shows that this configuration will double the capacity while maintaining availability. Introducing the load rebalancing feature, where when one unit fails, the remaining unit takes over the entire load, will improve reliability.

4.3 Combinations of the Basic Configurations/Templates

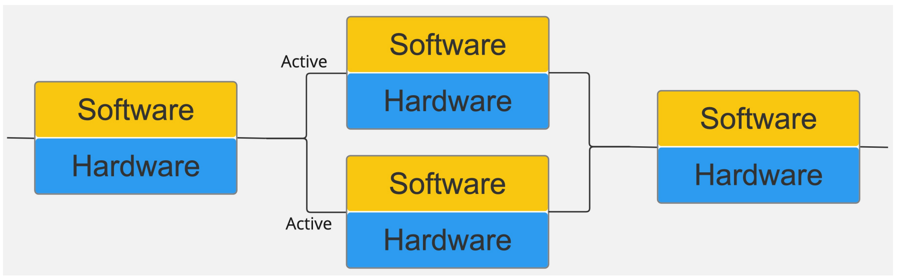


Fig. 8: A-C-A Configuration: GRED for two single-units and one two-unit active-active configuration

Each template requires specific starting and ending points to construct a network of templates. The network begins at the Start node and ends at the End node. Complex systems may combine series and parallel configurations. The reliability of such systems can be calculated by breaking them down into more straightforward series and parallel components and then combining their reliabilities using appropriate formulas. Sorting them topologically simplifies the metrics calculation. Figure 8 shows an example with three configurations: 2 single-unit configurations, a two-unit active-active load-sharing configuration. These configurations are connected in series. Reliability metrics are already calculated for each template, allowing us to add them to determine the system failure rate. For example:

$$\lambda_{sys} = (\lambda_{sw_A} + \lambda_{hw_A}) + (\lambda_{sw_B} + \lambda_{hw_B}) + (\lambda_{sw_C} + \lambda_{hw_C}) \tag{34}$$

Other metrics can be calculated similarly. For the availability of the entire system, we can use the sum of the unavailability of each configuration and subtract it from 1:

Table 8: FUSION Output Metrics Calculation Formulas for Configuration A-C-A

Metric	Formulas		
	Hardware	Software	System
Failure Rate	$\lambda_{hw_{all}} = 2\lambda_{hw_a} + \lambda_{hw_b}$	$\lambda_{sw_{all}} = 2\lambda_{sw_a} + \lambda_{sw_b}$	$\lambda_{sys_{all}} = 2\lambda_{sys_a} + \lambda_{sys_b}$
MTTF	$\frac{1}{\lambda_{hw_{all}}}$	$\frac{1}{\lambda_{sw_{all}}}$	$\frac{1}{\lambda_{sys_{all}}}$
Reliability	$e^{-\lambda_{hw_{all}} t}, t = 1$	$e^{-\lambda_{sw_{all}} t}, t = 1$	$e^{-\lambda_{sys_{all}} t}, t = 1$
MTBF	$\frac{MTTF_{hw_{all}}}{A_{hw_{all}}}$	$\frac{MTTF_{sw_{all}}}{A_{sw_{all}}}$	$\frac{MTTF_{sys}}{A_{sys}}$
MTTR [a]	$(\frac{2\lambda_{hw_a}}{\mu_{hw_a}} + \frac{\lambda_{hw_b}}{\mu_{hw_b}})/\lambda_{hw_{all}}$	$(\frac{2\lambda_{sw_a}}{\mu_{sw_a}} + \frac{\lambda_{sw_b}}{\mu_{sw_b}})/\lambda_{sw_{all}}$	$(\frac{\lambda_{hw_{all}}}{\mu_{hw_{all}}} + \frac{\lambda_{sw_{all}}}{\mu_{sw_{all}}})/\lambda_{sys_{all}}$
Availability	$2 - 2A_{hw_a} - A_{hw_b}$	$2 - 2A_{sw_a} - A_{sw_b}$	$2 - 2A_{sys_a} - A_{sys_b}$

[*] Must be adjusted for the year
[a] μ is the repair rate and is measured in **repairs per unit time**

Table 9: FUSION Outputs for Configuration A-C-A

Metric	Hardware	Software	System (SW + HW)
Failure rate	0.60	2.40	3.00
MTTF	1.6667	0.4167	0.3333
Reliability	0.54.9%	9.1%	5.0%
MTBF	1.6670	0.4168	0.3335
MTTR	240	120	144
Availability	99.979%	99.959%	99.938%

$$A = 1 - \sum(1 - A_i) \tag{35}$$

Table 8 provides the formulas for the reliability and availability metrics. Table 9 shows the outputs for the sample configurations. FUSION will automatically calculate for any combinations of the templates.

4.4 Possible other configuration options for future addition

We have selected three basic practical configurations for the Phase I implementation. However, many other configurations are appropriate for some users. We will provide a sample list of cases for future implementation.

- A two-unit active-active load-sharing system where:
  - When one unit fails, the remaining unit takes over the entire load, and the system continues to function until the second unit fails.

  - Extend from two units to multiple units.

- A two-unit parallel system.
- A two-unit active-standby system where:
  - The standby unit is powered up and ready, providing near-instantaneous response time. It is called a hot standby unit.
  - Include an imperfect switching case.
- Include an impact of interaction between software and hardware.

5 Conclusion

In this chapter, we introduced FUSION, an innovative cloud-based tool designed to integrate software and hardware models for analyzing defects and failures, providing a unified approach to system reliability. FUSION effectively addresses a critical gap in reliability modeling for the operational phase of software by aligning software reliability models with established hardware reliability models. Our research demonstrates that software failure rates during operation can be accurately modeled as segmented straight lines. Equipped with advanced analytics and a graphical editor, FUSION facilitates the creation of detailed reliability block diagrams, thereby expediting system reliability assessments and enabling targeted improvements. By allowing practitioners to enhance overall system performance and reliability, FUSION significantly improves customer satisfaction. This comprehensive tool represents a considerable advancement in the field of reliability engineering, paving the way for more robust and reliable integrated systems.

Acknowledgements. This research is supported by the National Science Foundation (NSF) Small Business Technology Transfer (STTR) program under Award Number 2348264.

Appendix A A Summary of Software Reliability Models

Many software reliability models based on the non-homogeneous Poisson processes (NHPP) [21–43] have been developed using the fault intensity rate function and the mean value functions within a controlled testing environment to estimate reliability metrics, including the number of residual faults, failure rate, and software reliability. Generally, these models are applied to software testing data and used to make predictions about software failures and reliability in the field under the assumption that the field environment and the software development environment are the same. In other words, the common underlying assumption of such models is that the operating and development environments are similar. However, operating environments in software are often quite different [9, 44]. The randomness of the operating environments [9, 45] affects the entire system's failure, particularly software failure.

Estimating software reliability in the field is essential yet challenging. Usually, software reliability models are applied to system test data to estimate the software's

Table A1: A summary of SRGMs for non-random operating environments [2, 21]

Model	$m(t)$
Goel-Okumoto (G-O)	$a(1-e^{-bt})$
Delayed S-shaped	$a(1-(1+bt)e^{-bt})$
Inflection S-shaped	$\frac{a(1-e^{-bt})}{1+\beta e^{-bt}}$
Yamada Imperfect debugging 1	$\frac{ab}{\alpha+b}(e^{\alpha t}-e^{-bt})$
Yamada Imperfect debugging 2	$a(1-e^{-bt})(1-\frac{\alpha}{b})+\alpha at$
Pham Inflexion	$N\left(1-\frac{1}{\left(\frac{\beta+e^{bt}}{1+\beta}\right)^{\frac{a}{b}}}\right)$
PNZ model	$\frac{a}{1+\beta e^{-bt}}\left((1-e^{-bt})(1-\frac{\alpha}{b})+\alpha t\right)$
Pham-Zhang model	$\frac{1}{1+\beta e^{-bt}}\left((c+a)(1-e^{-bt})-\frac{ab}{b-\alpha}(e^{-\alpha t}-e^{-bt})\right)$
Dependent-parameter model	$\alpha(1+\gamma t)(\gamma t-1+e^{-\gamma t})$
Dependent-parameter model with $m(t) \neq 0$	$m_0\left(\frac{\gamma t+1}{\gamma t_0+1}\right)e^{-\gamma(t-t_0)}+\alpha(1+\gamma t)(\gamma t-1+(1-\gamma t_0)e^{-\gamma(t-t_0)})$
Pham-Zhang IFD	$a-ae^{-bt}(1+(b+d)t+bdt^2)$
Zhang-Teng-Pham model	$\frac{a}{p-\beta}\left(1-\left(\frac{(1+\alpha)e^{-bt}}{a+\alpha e^{-bt}}\right)^{\frac{c}{b}(p-\beta)}\right)$
PZ-coverage	$a(1+\alpha t-\frac{bt+1}{e^{bt}})-\frac{a\alpha(1+bt)}{be^{bt+1}}\times\left(\ln(bt+1)+\sum_{i=0}^{\infty}\frac{(1+bt)^{i+1}-1}{(i+1)!(i+1)}\right)$
Dependent failure	$\frac{a}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bt}}\right)^{a}}$

failure rate in user environments [9]. Zhang and Pham [9] demonstrated a practical methodology to predict the field failure rate of software by analyzing system test data and field data of earlier releases using software reliability growth models (SRGMs). Teng and Pham [44] discussed a generalized model that captures the uncertainty of the environment and its effects on the software failure rate. Other researchers [45–50] have developed software reliability models that consider the uncertainty of the operating environment. Tables A1 and A2 summarize some standard NHPP software reliability models of the mean value function that can determine the expected number of residual faults after the testing phase for non-random and random operating environments, respectively.

Appendix B Linear Defect Prediction Model (LPM)

B.1 PART I: Finding inflection points and multiple curves

Input data

- (x_i, y_i) $\forall 0 \leq i \leq p$ is the cumulative number of defects found
- (x_0, y_0) and (x_p, y_p) represent the start and end weeks of the entire defect data, respectively.
- i is an index for the current week up to the end week
- j is an index for the week up to the current week i

Table A2: A summary of SRGMs for random operating environments [21]

Model	$m(t)$
Pham (Vtub) model	$N\left(1-\left(\frac{\beta}{\beta+d^{b}-1}\right)^{\alpha}\right)$
Logistic fault detection model	$N\left(1-\frac{\beta}{\beta+\left(\frac{c}{b}\right)\ln\left(\frac{a+e^{bt}}{1+a}\right)}\right)^{\alpha}$
Loglog uncertainty model	$N\left(1-\frac{\beta}{\beta+a^{t^{b}}-1}\right)^{\alpha}$
Pham Inflexion	$N\left(1-\frac{1}{\left(\frac{\beta+e^{bt}}{1+\beta}\right)^{\frac{a}{b}}}\right)$
Testing coverage uncertainty	$N\left(1-\frac{\beta}{\beta-\frac{a}{b}\ln\left(\frac{(1+c)e^{-bt}}{1+ce^{-bt}}\right)}\right)$
SRGM dependent failures	$N\left(1-\left(\frac{\beta}{\alpha+bt-\ln(bt+1)}\right)^{\alpha}\right)$

In line 9, we calculate the sum of squares of the difference between the predicted value and actual value. In line 11, we have a decision to identify an inflection point by comparing the last two weeks of SSQ values. In line 13, Check the relative changes of SSQ_i and SSQ_{i-1} to SSQ_{i-2} values, respectively . In Line 14 If both are greater than an accuracy threshold, default = 10%, then an inflection point is found at the week $i-2$. And continue the i loop with $x_s = x_{i-2}$ for the next inflection point

A set of multiple curves with start and end weeks and slope for each curve. Refer to a sample output in Table B3.

Table B3: A sample output of LPM Part I

Curve No.	Start	End	Slope
1	4/15/2019	5/6/2019	3.33
2	5/6/2019	5/27/2019	2.33
3	5/27/2019	6/17/2019	7.00
4	6/17/2019	7/8/2019	5.00
5	7/8/2019	7/29/2019	8.33
6	7/29/2019	9/2/2019	5.80
7	9/2/2019	9/23/2019	5.33
8	9/23/2019	11/25/2019	2.00

B.2 PART II: Adjusting the multiple curves based on the relative change in the slope

Algorithm 2 generates a final set of inflection points and slopes using the output of the LPM Part I. Its input is a set of multiple curves with start and end weeks and slope for each curve. Refer to a sample output in Table B4. Figures B1 and B2 demonstrate

Algorithm 1 Finding inflection points and multiple curves

```
procedure LPM($(x_i, y_i) \quad \forall 0 \leq i \leq p$)
    for $i = x_s, i \leq x_p, i++$ do                ▷ Start the loop at $x_s, y_s$ to the end week
        if $i \geq x_s + 3$ then
            $slope = \frac{y_i - y_s}{x_i - x_s}$
            for $j = x_s, j \leq i, j++$ do
                $\hat{y_j} = slope \times (x_j - x_s) + y_s$        ▷ Calculate the predicted value
                $SSQ_j = (y_j - \hat{y_j})^2$        ▷ This is a measure of goodness of fit
            end for
            $SSQ_i = \sum_{j=x_s}^{i} \frac{SSQ_j}{(i - x_s)}$        ▷ Goodness of fit for the week $x_s$ to week $i$
            if $i \geq x_s + 5$ then ▷ Check if we have the last two weeks of SSQ values
                $\Delta SSQ_1 = \frac{SSQ_{i-2} - SSQ_{i-1}}{SSQ_{i-2}}$
                $\Delta SSQ_2 = \frac{SSQ_{i-2} - SSQ_i}{SSQ_{i-2}}$
                if $i \geq x_s + 5$ then
                    if ($\Delta SSQ_1 > 0.1$) AND ($\Delta SSQ_2 > 0.1$) then
                        IP = Week $i - 2$        ▷ IP is the Inflection Point
                    else
                        $x_s = x_{i-2}$
                    end if
                end if
            end if
        end if
    end for
    return IP
end procedure
```

the predicted cumulative defects and weekly defects, respectively, using the LPM Part I and II with actual data.

Appendix C Algorithm for Software Failure Prediction

C.1 Input Data

Fig. C3 shows the required input data. It includes customer defect predicted curve (i.e., output of LPM) and the software failure data (i.e., critical defects).

C.2 Analytics for software failure prediction

C.2.1 Transformation function

- The cumulative predicted defect curve (x, y) is transformed into the cumulative failure curve (u, v) using the parameters, α, β, and γ:

$$u = \alpha + \beta x \tag{C1}$$

Table B4: A sample table illustrating the LPM Part II procedure

	Curve	Start	End	Slope	Rel. Change	Action	SSQ	Rel. Change
Original Curve	1	15/04/2019	06/05/2019	3.33			7.88	NA
	2	06/05/2019	27/05/2019	2.33	0.30			
	3	27/05/2019	17/06/2019	7.00	2.00			
	4	17/06/2019	08/07/2019	5.00	0.29			
	5	08/07/2019	29/07/2019	8.33	0.67			
	6	29/07/2019	02/09/2019	5.80	0.30			
	7	02/09/2019	23/09/2019	5.33	0.08	merge		
	8.00	23/09/2019	25/11/2019	2.00	0.62			
Iteration 1	1	15/04/2019	06/05/2019	3.33			8.15	0.03
	2	06/05/2019	27/05/2019	2.33	0.30			
	3	27/05/2019	17/06/2019	7.00	2.00			
	4	17/06/2019	08/07/2019	5.00	0.29	merge		
	5	08/07/2019	29/07/2019	8.33	0.67			
	6	29/07/2019	23/09/2019	5.63	0.33			
	7.00	23/09/2019	25/11/2019	2.00	0.64			
Iteration 2	1	15/04/2019	06/05/2019	3.33			10.27	0.26
	2	06/05/2019	27/05/2019	2.33	0.30	merge		
	3	27/05/2019	08/07/2019	6.00	1.57			
	4	08/07/2019	29/07/2019	8.33	0.39			
	5	29/07/2019	23/09/2019	5.63	0.33			
	6.00	23/09/2019	25/11/2019	2.00	0.64			
Iteration 3	1	15/04/2019	27/05/2019	2.83			10.30	0.003
	2	27/05/2019	08/07/2019	6.00	1.12			
	3	08/07/2019	29/07/2019	8.33	0.39			
	4	29/07/2019	23/09/2019	5.63	0.33	merge		
	5.00	23/09/2019	25/11/2019	2.00	0.64			
Iteration 4	1	15/04/2019	27/05/2019	2.83			16.68	0.62
	2	27/05/2019	08/07/2019	6.00	1.12			
	3	08/07/2019	23/09/2019	6.36	0.06	merge		
	4.00	23/09/2019	25/11/2019	2.00	0.69			
Iteration 5	1	15/04/2019	27/05/2019	2.83			14.35	-0.14
	2	27/05/2019	23/09/2019	6.23	1.20			
	3.00	23/09/2019	25/11/2019	2.00	0.68	merge		
Iteration 6	1	15/04/2019	27/05/2019	2.83			82.38	4.74
	2.00	27/05/2019	25/11/2019	4.77	0.68	merge		

$$v = \gamma y \tag{C2}$$

- Parameter interpretations
  - α: Horizontal shift (right if positive, left if negative, no shift if zero).
  - β: Scaling factor (delay if greater than 1, acceleration if less than 1, no change if 1).
  - γ: Vertical scaling factor (doubles height if 2, reduces to 50% if 0.5, no change if 1).
- A non-linear optimization problem is solved to find the best values of α, β, and γ that minimize the difference between the actual and transformed failure curves.

Algorithm 2 Adjusting the multiple curves based on the relative change in the slope

```
1: Create a table of start & end dates and slope for each segmented line of the original curve (i.e., the output of the LPM Part I).
2: Calculate SSQ, which is the square root of the sum of the squares of the distance between the actual value and the predicted curve.
3: Calculate the relative change in the slope value from one segment to the next.
4: Identify the smallest relative change that will be merged.
5: Merge the two segments identified: (1) The start date is replaced with the previous one. (2) Calculate the slope for the new segment based on the new start & end dates.
6: Create a new table with the merged segment. See Table 2.
7: Calculate SSQ for the new curve.
8: Calculate the relative change in SSQ from the previous curve.
9: Repeat steps 2 through 7 until the relative change in SSQ is greater than 1 (a new parameter).
10: The final predicted curve is the previous curve.
11: The expected output for troubleshooting is a set of tables with each iteration and the original and final predicted curve, as shown in the sample tab.
```

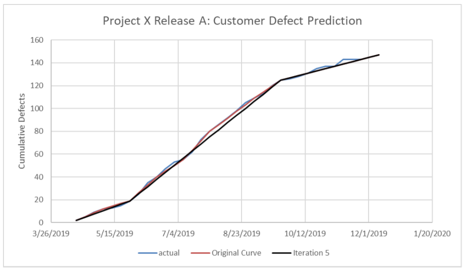


Fig. B1: Predicted cumulative curves generated by LPM Part I and II with actual data.

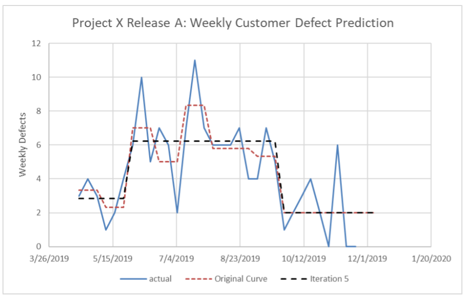


Fig. B2: Predicted weekly curves generated by LPM Part I and II with actual data.

C.2.2 Algorithm

The algorithm optimizes the parameters α, β, and γ to best fit the predicted failure curve to the actual one. The objective function is based on the absolute difference between predicted defects and actual failures over the last 10 (default) weeks.

Objective function

To calculate the objective function, follow these steps:

- Calculate transformed values (u, v) using the equations (C1) and (C2) for given values of α, β, and γ.

Table C5: Sample calculation of the objective function b.2 Setting parameter rages

Customer		Failure curve					
Week	Predicted curve	Week	Predicted	Week	Interpolated	Actual	Absolute
x	y	u	v	x_act	v_act	y_act	difference
1	2	5.35	0.18	1			
2	5	6.20	0.42	2			
3	7	7.05	0.66	3			
4	10	7.90	0.90	4	0.00	1	1.00
5	12	8.75	1.14	5	0.00	1	1.00
6	15	9.60	1.38	6	0.37	1	0.63
7	21	10.45	1.94	7	0.65	1	0.35
8	27	11.30	2.50	8	0.93	1	0.07
9	33	12.15	3.07	9	1.21	2	0.79
10	39	13.00	3.63	10	1.64	2	0.36
11	46	13.85	4.19	11	2.31	2	0.31
12	52	14.70	4.75	12	2.97	3	0.03
13	58	15.55	5.32	13	3.63	3	0.63
14	64	16.40	5.88	14	4.29	5	0.71
15	70	17.25	6.44	15	4.95	5	0.05
16	76	18.10	7.00	16	5.61	6	0.39
17	82	18.95	7.56	17	6.27	6	0.27
18	88	19.80	8.13	18	6.94	6	0.94
19	94	20.65	8.69	19	7.60	7	0.60
20	101	21.50	9.25	20	8.26	8	0.26

- The actual failure curve (x_{act}, y_{act}) remains on the original (x, y) scale. Calculate v_{act}, for $u = x_{act}$ using interpolation:

$$v_{act} = v_1 + \frac{v_2 - v_1}{u_2 - u_1} \times (x_{act} - u_1) \tag{C3}$$

- Calculate the sum of the absolute differences (SSQ) between v_{act}, and y_{act}:

$$SSQ = \sum |v_{act} - y_{act}| \tag{C4}$$

Note: Table C5 demonstrates a sample calculation of the objective function with sample parameters $\alpha = 4.5$, $\beta = 0.85$, and $\gamma = 0.092$. The parameter ranges are as follows:

- α: 5 to 0 with a decrement of 0.5
- β: 0.7 to 1.1 with an increment of 0.05
- γ: 0.86 to 1.2 with an increment of 0.

Solving the non-linear optimization problem

- The algorithm iterates over three levels:
  - Level 1: Iterate over γ from 0.86 to 1.2 with an increment of 0.02.

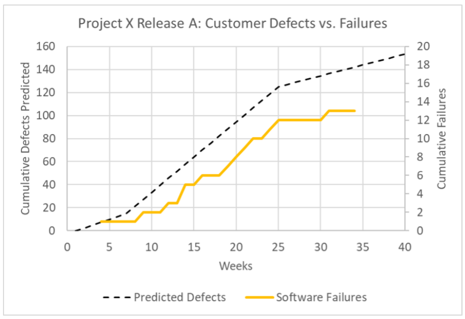


Fig. C3: Sample data for customer defects and software failures

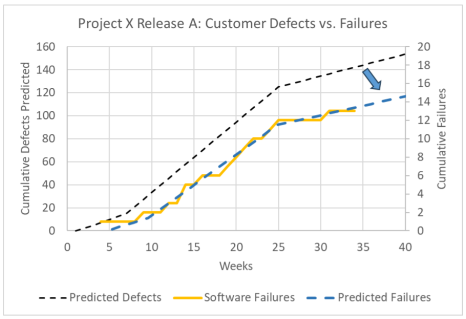


Fig. C4: Sample data for predicted software failure curve

  - Level 2: For each value of γ, iterate over β from 0.7 to 1.1 with an increment of 0.05.
  - Level 3: For each combination of γ and β, iterate over α from 5 to 0 with a decrement of 0.5.

- Calculate SSQ for each combination and track the minimum SSQ value:
  - For a fixed value of γ, iterate through α and β. Terminate Level 3 if the SSQ exceeds the previous value.
  - Continue Level 2 until the minimum SSQ for the current γ is found.
  - Repeat the process for γ until the global minimum SSQ is identified.

This algorithm provides a methodical approach to determining the best-fit parameters that minimize discrepancies between the predicted and actual software failure curves (see Figure C4).

References

[1] Dan, M.: The business model of "Software-As-A-Service". In: Proceedings - 2007 IEEE International Conference on Services Computing, SCC 2007 (2007). https://doi.org/10.1109/SCC.2007.118

[2] Pham, H.: Software Reliability. Springer, Berlin, Heidelberg (1999)

[3] O'Connor, P.D.T.: Software Reliability Engineering , John D. Musa, McGraw-Hill, 1999. Number of pages: 391. Quality and Reliability Engineering International **16**(5) (2000)

[4] Lyu, M.R.: Handbook of Software Reliability Engineering. McGraw-Hill, Inc., USA (1996)

[5] Hanagal, D.D., Bhalerao, N.N.: Literature Survey in Software Reliability Growth Models. In: Software Reliability Growth Models, pp. 13–26. Springer, Singapore (2021).

[6] Mijumbi, R., Okumoto, K., Asthana, A.: Software Reliability Assurance in Practice. In: Wiley Encyclopedia of Electrical and Electronics Engineering, (2019). https://doi.org/10.1002/047134608x.w6952.pub2

[7] Okumoto, K.: Early Software Defect Prediction: Right-Shifting Software Effort Data into a Defect Curve. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 43–48 (2022). https://doi.org/10.1109/ISSREW55968.2022.00037

[8] Okumoto, K.: Digital Engineering-driven Software Quality Assurance-as-a-Service. In: 28th ISSAT International Conference on Reliability and Quality in Design, RQD 2023 (2023)

[9] Zhang, X., Pham, H.: Software field failure rate prediction before software deployment. Journal of Systems and Software **79**(3) (2006) https://doi.org/10.1016/j.jss.2005.05.015

[10] Okumoto, K.: Experience Report: Practical Software Availability Prediction in Telecommunication Industry. In: Proceedings - International Symposium on Software Reliability Engineering, ISSRE (2016). https://doi.org/10.1109/ISSRE.2016.19

[11] Zhu, M., Pham, H.: A novel system reliability modeling of hardware, software, and interactions of hardware and software. Mathematics **7**(11) (2019) https://doi.org/10.3390/math7111049

[12] O'Connor, P.D.T., Kleyner, A.: Practical Reliability Engineering: Fifth Edition, (2011). https://doi.org/10.1002/9781119961260

[13] US Department of Defense: Military Handbook MIL-HDBK-217E, Reliability Prediction of Electronic Equipment, (1986)

[14] Ryan, T.P., Meeker, W.Q.: System Reliability Theory: Models, Statistical Methods, and Applications, Second Edition. Journal of Quality Technology **37**(1) (2005) https://doi.org/10.1080/00224065.2005.11980303

[15] Nelson, W., Meeker, W.Q., Escobar, L.A.: Statistical Methods for Reliability Data. Technometrics **40**(3) (1998) https://doi.org/10.2307/1271181

[16] Kleyner, A., Volovoi, V.: Application of Petri nets to reliability prediction of occupant safety systems with partial detection and repair. Reliability Engineering and System Safety **95**(6) (2010) https://doi.org/10.1016/j.ress.2010.01.008

[17] Birolini, A.: Reliability Engineering: Theory and Practice, Seventh Edition, (2014). https://doi.org/10.1007/978-3-642-39535-2

[18] Jackson, Y., Tabbagh, P., Gibson, P., Seglie, E.: The new department of defense (DoD) guide for achieving and assessing RAM. In: Proceedings - Annual Reliability and Maintainability Symposium (2005). https://doi.org/10.1109/rams.2005.1408329

[19] Relyence: A Deep Dive into System Modeling Using Reliability Block Diagram (RBD) Analysis (2021)

[20] Relyence: A Deep Dive into System Modeling Using Reliability Block Diagram (RBD) Analysis. Technical report (2021). https://relyence.com/wp-content/uploads/2021/09/A-Deep-Dive-into-System-Modeling-using-Reliability-BlockDiagram-RBD-Analysis.pdf

[21] Pham, H., Teng, X.: Software Reliability Modeling and Prediction. In: Springer Handbooks, (2023).

[22] Goel, A.L., Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. IEEE Transactions on Reliability **R-28**(3) (1979) https://doi.org/10.1109/TR.1979.5220566

[23] Yamada, S., Ohba, M., Osaki, S.: S-Shaped Reliability Growth Modeling for Software Error Detection. IEEE Transactions on Reliability **R-32**(5), 475–484 (1983) https://doi.org/10.1109/TR.1983.5221735

[24] Ohba, M.: Inflection S-Shaped Software Reliability Growth Model, (1984).

[25] Yamada, S., Tokuno, K., Osaki, S.: Imperfect debugging models with fault introduction rate for software reliability assessment. International Journal of Systems Science **23**(12) (1992) https://doi.org/10.1080/00207729208949452

[26] Pham, H., Zhang, X.: An NHPP software reliability model and its comparison. International Journal of Reliability, Quality and Safety Engineering **4**(3) (1997) https://doi.org/10.1142/S0218539397000199

[27] Pham, H., Nordmann, L., Zhang, X.: A general imperfect-software-debugging model with s-shaped fault-detection rate. IEEE Transactions on Reliability **48**(2) (1999) https://doi.org/10.1109/24.784276

[28] Zhang, X., Teng, X., Pham, H.: Considering Fault Removal Efficiency in Software Reliability Assessment. IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans. **33**(1) (2003) https://doi.org/10.1109/TSMCA.2003.812597

[29] Pham, H., Zhang, X.: NHPP software reliability and cost models with testing coverage. European Journal of Operational Research **145**(2) (2003) https://doi.org/10.1016/S0377-2217(02)00181-9

[30] Hwang, S., Pham, H.: Quasi-renewal time-delay fault-removal consideration in software reliability modeling. In: IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans, vol. 39 (2009). https://doi.org/10.1109/TSMCA.2008.2007982

[31] Pham, H.: An imperfect-debugging fault-detection dependent-parameter software. International Journal of Automation and Computing **4**(4) (2007) https://doi.org/10.1007/s11633-007-0325-8

[32] Kapur, P.K., Pham, H., Anand, S., Yadav, K.: A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Transactions on Reliability **60**(1) (2011) https://doi.org/10.1109/TR.2010.2103590

[33] Li, Q., Pham, H.: A testing-coverage software reliability model considering fault removal efficiency and error generation. PLoS ONE **12**(7) (2017) https://doi.org/10.1371/journal.pone.0181524

[34] Singh, V.B., Sharma, M., Pham, H.: Entropy based software reliability analysis of multi-version open source software. IEEE Transactions on Software Engineering **44**(12) (2018) https://doi.org/10.1109/TSE.2017.2766070

[35] Zhu, M., Pham, H.: A multi-release software reliability modeling for open source software incorporating dependent fault detection process. Annals of Operations Research **269**(1-2) (2018) https://doi.org/10.1007/s10479-017-2556-6

[36] Pham, T., Pham, H.: A generalized software reliability model with stochastic fault-detection rate. Annals of Operations Research **277**(1) (2019) https://doi.org/10.1007/s10479-017-2486-3

[37] Zhu, M., Pham, H.: A software reliability model incorporating martingale process with gamma-distributed environmental factors. Annals of Operations Research (2018) https://doi.org/10.1007/s10479-018-2951-7

[38] Chatterjee, S., Shukla, A., Pham, H.: Modeling and analysis of software fault detectability and removability with time variant fault exposure ratio, fault removal efficiency, and change point. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability **233**(2) (2019) https://doi.org/10.1177/1748006X18772930

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

[40] Kim, Y.S., Song, K.Y., Pham, H., Chang, I.H.: A Software Reliability Model with Dependent Failure and Optimal Release Time. Symmetry **14**(2) (2022) https://doi.org/10.3390/sym14020343

[41] Li, Q., Pham, H.: Software reliability modeling incorporating fault detection and fault correction processes with testing coverage and fault amount dependency. Mathematics **10**(1) (2022) https://doi.org/10.3390/math10010060

[42] Zhu, M., Pham, H.: A generalized multiple environmental factors software reliability model with stochastic fault detection process. Annals of Operations Research **311**(1) (2022) https://doi.org/10.1007/s10479-020-03732-3

[43] Shrivastava, A.K., Sharma, R., Pham, H.: Software reliability and cost models with warranty and life cycle. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability **237**(1) (2023) https://doi.org/10.1177/1748006X221076273

[44] Teng, X., Pham, H.: A new methodology for predicting software reliability in the random field environments. IEEE Transactions on Reliability **55**(3), 458–468 (2006) https://doi.org/10.1109/TR.2006.879611

[45] Pham, H.: A software reliability model with Vtub-shaped fault-detection rate subject to operating environments. In: Proceedings - 19th ISSAT International Conference on Reliability and Quality in Design, RQD 2013 (2013)

[46] Pham, H.: A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. Optimization: Journal of Mathematical Programming and Operations Research **63**(10), 1481–1490 (2014) https://doi.org/10.1080/02331934.2013.854787

[47] Song, K.Y., Chang, I.H., Pham, H.: A three-parameter fault-detection software reliability model with the uncertainty of operating environments. Journal of Systems Science and Systems Engineering **26**(1) (2017) https://doi.org/10.1007/s11518-016-5322-4

[48] Lee, D.H., Chang, I.H., Pham, H.: Software Reliability Growth Model with Dependent Failures and Uncertain Operating Environments. Applied Sciences (Switzerland) **12**(23) (2022) https://doi.org/10.3390/app122312383

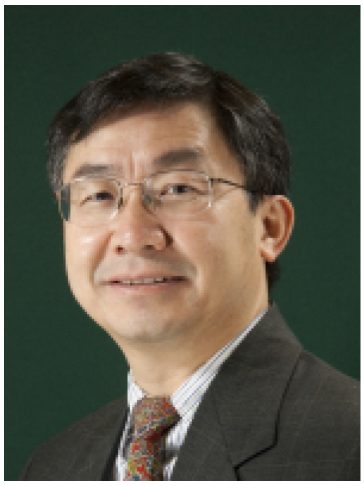
[49] Song, K., Kim, Y.S., Pham, H., Chang, I.H.: A Software Reliability Model Considering a Scale Parameter of the Uncertainty and a New Criterion. Mathematics (2024)

[50] Okumoto, K.: Design, Evaluation and Implementation of Algorithms for Predicting Software Defects and Failures During the Operational Phase. In: International Society of Science and Applied Technologies. ISSAT, ??? (2024). https://issatconferences.org/Abstracts/RQD/Content$_RQD/content$2024/46$.html$

Author Biographies

Kazuhira Okumoto

Dr. Kazuhira 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 RD. 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.



Hoang Pham

Hoang Pham is a Distinguished Professor in the Department of Industrial and Systems Engineering at Rutgers University, Piscataway. He is the Editor-in-Chief of the International Journal of Reliability, Quality and Safety Engineering and the Editor of the Springer Series in Reliability Engineering. He is a Fellow of the IEEE and IISE.