Advances in Reliability, Safety and Security, Part 6 – Kołowrocki, Magryta-Mut (eds.)

Advances in Reliability, Safety and Security

ESREL 2024
Monograph Book Series

Deep Neural Network Based Software Reliability Growth Model For Fault Prediction And Severity Assessment

Shikha Dwivedi, Neeraj Kumar Goyal

SCSQR, IIT Kharagpur, West Bengal, India

Abstract

Software reliability is a critical aspect of modern software development, and predicting faults is helpful in testing resource allocation and achieving reliability targets for software release. This paper introduces a novel Deep Neural Network-based Software Reliability Growth Model (DNN-based SRGM) that not only forecasts detected and corrected faults but also predict the fault severity. The proposed model utilizes a Bi-directional GRU (BiGRU) architecture to capture the fault detection process (FCP), fault correction process (FCP) and the severity of corrected faults (minor, major, and critical). The model is validated using two real datasets showcasing superior performance compared to existing models, as demonstrated through results across six evaluation criteria on both the datasets. The model is also compared with existing parametric and non-parametric models, demonstrating predictive accuracy, and the ability to address challenges related to fault correction and severity considerations. Results, particularly for severity prediction, are presented and evaluated against actual faults using error metrics. The model contributes to improved software reliability by providing software development teams with more accurate and informative insights into testing outcomes for prioritizing resources and addressing critical issues proactively.

Keywords: software reliability, ANN, fault detection, fault correction, severity

1. Introduction

The relentless pace of technological innovation presents an exciting paradox. While new advancements emerge at breakneck speed, the underlying software must remain reliable to underpin this progress. In this rapidly evolving landscape, software reliability becomes more critical aspect. The consequences can be catastrophic, highlighting the vital need for robust and dependable software.

To address this growing concern, researchers have developed various software reliability growth models (SRGMs) (Dhaka & Nijhawan, 2022; Pradhan et al., 2022; Singhal et al., 2023; Yang et al., 2016). These models attempt to predict and improve software reliability by analyzing collected data on faults. Traditionally, parameter models based on assumptions like Poisson processes or Bayes processes have dominated the field (Huang et al., 2022). However, these models often struggle with inflexibility and their pre-defined assumptions may not always hold true in complex real-world scenarios, leading to inaccurate predictions. Additionally, the reliance on specific estimation methods introduces further limitations, rendering the results susceptible to variations. Seeking to overcome these limitations, researchers explored non-parameter models like neural networks (NNs) (Bisi & Goyal, 2016; Pandey & Goyal, 2010; Karunanithi et al., 1991). These models represent a significant shift, as they learn from data without relying on pre-defined assumptions. This allows them to adapt to diverse software scenarios and capture intricate relationships that might be missed by parameter models. However, they faced their own challenges, including **overfitting** and **complex parameter tuning,** which hampered their effectiveness.

The recent surge in **deep neural networks (DNNs)** presents a promising new path for overcoming these limitations. Deep neural networks (DNNs), with their multi-layered architecture, exhibit superior capabilities in learning complex patterns and hidden features within data. This presents a unique opportunity to address the limitations of existing models and achieve more accurate and generalizable software reliability predictions. However, even current DNN-based SRGMs (C. Li et al., 2022; Wang & Zhang, 2018) have limitations. They often

neglect key factors like fault correction, severity and the influence of external factors on software reliability. To address these limitations, this research introduces a pioneering DNN-based SRGM that harnesses the capabilities of deep learning while explicitly incorporating fault correction and severity into the prediction process. Notably, the model utilizes Bidirectional Gated Recurrent Units (BiGRUs) to enhance its understanding of temporal dependencies in fault correction processes. By categorizing faults based on their impact, the model provides a multi-level severity assessment, improving its ability to differentiate between critical and non-critical issues. This integration empowers software developers to prioritize their efforts effectively, considering the actual impact of faults on system performance and user experience. To validate the effectiveness of our model, we utilize real-world datasets and employ six evaluation criteria, including Mean Squared Error (MSE), bias, variation, Akaike Information Criterion (AIC), and Adjusted R^2. Through rigorous validation using existing parametric and non-parametric model, our DNN-based SRGM showcases superior performance, demonstrating its potential to advance non-parametric models by integrating fault correction mechanisms and severity assessment into their predictive frameworks.

The remaining paper includes the DNN based SRGM considering fault correction and severity in Section 2. Next, proposed methodology has been discussed in Section 3, followed by numerical example in Section 4 to validate the model and at last, the work has been concluded in Section 5.

2. DNN based SRGM considering fault correction process and severity

We propose a **deep learning model based on Bidirectional Gated Recurrent Units (Bi-GRUs)** to predict faults. This architecture effectively captures the complex relationships between fault detection, correction, severity, and their evolution across testing time.

2.1. Fault detection and correction process

The fault removal process in software involves an iterative process of detection, correction, and retesting. Timely identification and rectification of faults minimize their impact on system functionality and reliability. Traditional models may struggle to capture the intricate dependencies between successive fault correction cycles. In the context of traditional Software Reliability Growth Models (SRGMs), fault detection and correction are typically predicted using parametric approaches (Hsu et al., 2011; Peng et al., 2018; Pradhan et al., 2022), where mathematical models attempt to fit historical fault data. However, these models exhibit drawbacks such as assumption and an inability to adapt to evolving software environments. The proposed model integrates these challenges by leveraging historical data, enabling the model to learn patterns and relationships between fault detection, correction, and severity. The DNN predicts future detected and corrected faults along with their severity, crucial for resource allocation and reliability estimation. Bidirectional Gated Recurrent Units (Bi-GRUs) operates in both forward and backward directions simultaneously, excel in capturing temporal dependencies within fault correction processes, enhancing predictions. GRUs' selective information update and retention enable pattern learning, adapting to variations over time. Furthermore, Bi-GRUs adeptly handle the dynamic nature of fault correction processes, accommodating variable-length sequences by processing them bidirectionally.

2.2. Fault severity

Traditionally, SRGMs primarily focus on the number of faults detected, neglecting the crucial aspect of **fault severity.** This subsection delves into the incorporation of Fault Severity considerations within the DNN-based SRGM. In the proposed model, faults are systematically categorized based on their impact on the system. This categorization distinguishes faults into three levels: *minor*, *major*, and *critical*. Each category represents a different degree of severity, based on the potential impact on system functionality and user experience. **Minor** faults cause minimal disruption, while **major** ones affect functionality significantly. **Critical** faults pose serious risks and demand immediate attention. The model is trained to categorize faults based on their impact, providing a multi-level severity assessment (S_t). S_t denotes the severity of corrected faults at time t, representing a 3-dimensional vector (n_{1t}, n_{2t}, n_{3t}), where:

n_{1t}: number of minor faults corrected in time t

n_{2t}: number of major faults corrected in time t

n_{3t}: number of critical faults corrected in time t

By considering severity levels, the DNN-based SRGM enhances its capacity to distinguish between critical and non-critical issues, empowering software developers to prioritize their efforts effectively. It also mitigates risks and provides a realistic assessment of software reliability.

2.3. Problem definition

- Variables D_t, C_t, S_t represents the cumulative faults detected, corrected, and their severity at testing time t.
- Fault removal process $R_t = [D_t, C_t, S_t]$ integrates Fault Detection Process (FDP), Fault Correction Process (FCP), and fault severity for each period t.
- In each testing period t, historical data $\{R_1, R_2, \ldots, R_t\}$ is collected in the dataset δ_t.
- Develop a stepwise prediction model to estimate $R_{t+1} = [D_{t+1}, C_{t+1}, S_{t+1}]$ at the end of time t using historical dataset $\{R_1, R_2, \ldots, R_t\}$.
- Sliding window approach is used for neural network training and prediction, generating R_{t+1} from historical data $\{R_{t-k+1}, R_{t-k+}2, \ldots, R_t\}$, where k is the window size.
- The sliding window data, e.g., $\{R_1, R_2, R_3\}$, is updated over time, allowing predictions based on the most recent historical data.
- Choice of k depends on the total number of periods, with shorter testing time favoring smaller k and longer testing time using a larger k for accurate neural network predictions.

3. Proposed methodology

3.1. Model architecture

3.1.1. Input Layer:

The input layer takes as input the historical software testing data $R_t = [D_t, C_t, S_t]$ for each testing period t. The input layer receives the testing data within the sliding window of size k. This data will be a tensor of shape (k, 3), where k represents the window size and 3 represents the three features: D_t (cumulative detected faults), C_t (cumulative corrected faults), and S_t (severity).

3.1.2. Bi-GRU Layers:

Two stacked Bi-GRU layers are employed to process the sequences. Each layer comprises both forward and backward GRUs that analyze the information flow in both directions, enabling the model to capture temporal dependencies and context within the sequence.

3.1.3. Output Layer:

The output layer for R_{t+1} would predict for the next testing period, including: Cumulative number of faults detected (D_{t+1}) and faults categorized by severity (S_{t+1}), representing the expected distribution of faults (n_{1t+1}, n_{2t+1}, n_{3t+1}), that sum to the total number of expected corrected faults (C_t) in the next testing period.

3.1.4. Data Normalization

The collected testing dataset requires normalization before being fed into the neural network due to unevenly distributed values in the range of failure data. To mitigate this, Logarithmic encoding, as proposed by (Bisi and Goyal, 2016), is employed for this purpose. It scales all features within a specific range, typically 0 to 1, ensuring dimensional consistency and facilitating better learning by the network. This method transforms the input data (R_t) using the following formula:

$$X^* = \ln(1 + \beta x) \tag{1}$$

Here, β is the encoding parameter, x is input value and X^* is encoded value. The encoding parameter, β, plays a crucial role in scaling the data effectively. Its value is determined by considering maximum input value x_{max} and maximum value of the encoded input x^*_{max}.

$$\beta = -\frac{(ln(1-x^*_{max}))}{x_{max}} \tag{2}$$

The experiments done by (Bisi and Goyal, 2016) shows that choosing x^*_{max} ***within the range of 0.85 to 0.96*** for the logarithmic encoding leads to consistently lower **Mean Absolute Percentage Error (MAPE)** across diverse datasets. This indicates that scaling the data within this particular range allows the neural network to learn more effectively. By utilizing logarithmic encoding with a carefully chosen β and x^*_{max}, we ensure that the normalized data falls within the desired range while preserving valuable information about the original data distribution.

3.2. Network training and validation

To validate the efficacy of the DNN-based SRGM, rigorous training and validation processes are undertaken. We divide the pre-processed dataset into two parts: **training set** and **validation set**. The training set, representing the set of the data, used to teach the model the underlying patterns and relationships. The validation set, typically smaller, serves as an independent benchmark to assess the model's generalizability and prevent overfitting. The training phase involves presenting the neural network with historical testing data sequences, represented by sliding windows, and corresponding target values (e.g., Detected faults and future corrected faults categorized by severity). The network uses an Adam **optimizer** to adjust its internal parameters, gradually minimizing a mean squared error **loss function**. This process iteratively refines the network's ability to map input data patterns to the desired outputs. Training data is divided into smaller batches, and the network updates its parameters after processing each batch. The entire training dataset is passed through the network multiple times. Each epoch allows the network to learn more complex relationships within the data. The training process minimizes the loss function by adjusting the network weights, optimizing the model's ability to map past testing data to accurate predictions for the next period.

3.3. Prediction

Once our model has been trained and validated, we prepare testing data, ensuring its format aligns with the trained model's requirements like normalization. We pass the prepared data through the trained model, triggering its internal computations to generate predictions. The model outputs predictions for the next testing period, including:

- Cumulative Number of Faults Detected (Dt+1): This value helps assess the effectiveness of testing efforts and identify potential areas for improvement.
- Cumulative Number of Corrected Faults Categorized by Severity (St+1): By predicting the distribution of minor, major, and critical faults, we gain valuable insights into the potential impact of upcoming issues and can prioritize resources accordingly.

4. Numerical Example

4.1. Data collection

We have utilized two project datasets containing historical software testing data for model training and evaluation, including:

- Testing time (t)
- Cumulative number of faults detected (D_t)
- Cumulative number of faults corrected (C_t)
- Number of faults categorized by severity (n1, n2, n3 for minor, major, and critical respectively)

The First dataset originates from the Tomcat Maven product (https://issues.apache.org/jira/), as outlined in the Table 7 of appendix A. This dataset encompasses 70 weeks of software testing, revealing a total of 260 detected faults and 232 corrected faults. The severity is classified into categories such as minor, major, and critical faults. The second dataset is sourced from Apache Age (https://bz.apache.org/bugzilla/), spanning 105 weeks of testing with severity classifications denoted as minor, major, and high. This dataset records a total of 556 detected faults, of which 534 have been corrected. Since our model requires a consistent severity classification scheme (minor, major, and critical), we consider "high" faults from this dataset as equivalent to "critical" faults. The detailed information for this dataset is provided in Table 8 of Appendix A.

4.2. Model implementation

1. Divide the dataset into training (80%) and testing (20%) sets.
2. Preprocess the data, including logarithmic encoding for input features.
3. Train the Bi-GRU model with defined hyperparameters on the training set using MSE loss functions.
4. Monitor validation performance metrics (MSE, MAE) during training to avoid overfitting.
5. Use the trained model to predict D_{t+1}, C_{t+1} and S_{t+1} for each time period in the testing set.
6. Compare the predicted values with the actual values in the validation set.
7. Calculate relevant metrics like MSE, Bias Variation etc. for both D_{t+1} and C_{t+1} predictions, reflecting overall prediction accuracy.

4.3. Model Evaluation and Comparison:

The proposed DNN-based SRGM is evaluated against existing parametric and non-parametric models. This section presents a comparative analysis, highlighting the advantages of the neural network model in terms of prediction accuracy, adaptability, and its ability to address the specific challenges posed by fault correction and severity considerations.

4.3.1. Evaluation Criteria:

To assess the effectiveness of the proposed model compared to existing approaches, we employ six different error criteria:

- Mean Squared Error (MSE): Measures the average squared difference between predicted and actual values. Lower MSE indicates better prediction accuracy.
- Bias: Evaluates the systematic tendency of the model to under- or overestimate the actual values. Ideally, a model should exhibit minimal bias.
- Variance: Assesses the spread of the predictions around the average. Lower variance signifies higher consistency in the model's predictions.
- Akaike Information Criterion (AIC): A measure of model complexity and goodness-of-fit. Lower AIC scores indicate better models, balancing accuracy with complexity.
- Adjusted R-squared: An adjusted version of the R-squared statistic that accounts for the number of model parameters. Values closer to 1 indicate a better fit between the model and the data.

4.3.2. Comparative Models:

We compare our DNN-based SRGM with four existing models:

- Xiao Model (Xiao et al., 2020) and Hu Model (Hu et al., 2007): These are non-parametric models utilizing neural networks for predicting detected and corrected faults. While similar to our approach in using neural networks, they have not addressed severity classification.
- Dhaka Model (Dhaka and Nijhawan, 2022) and Li Model (Q. Li and Pham, 2017): These are statistical software reliability models designed for fault correction but not incorporated the added complexity of severity classification like proposed model.

Therefore, the comparison with these models will be limited to predicting detected and corrected faults (excluding severity) to ensure a fair comparison.

4.4. Performance analysis

4.4.1. Fault detection and Correction of the Datasets

This section analyses the performance of the proposed Deep Neural Network (DNN)-based Software Reliability Growth Model (SRGM) and compares it to existing models from the literature.

Evaluation Setup: Both datasets are pre-processed by dividing into sequences of length 5 ($k = 5$) and using Logarithmic encoding with an encoding parameter of 0.95 is applied for normalization. The model training employed 70% of the pre-processed data for each dataset. The core architecture utilized two Bi-directional GRU (BiGRU) layers, one with 64 units and the other with 32 units. Both layers used ReLU activation functions for efficient learning. To optimize the training process, the Adam optimizer was employed for 50 epochs and a batch size of 18 for Dataset 1, and 100 epochs with a batch size of 20 for Dataset 2. Mean squared error (MSE) served as the loss function, and early stopping was implemented to prevent overfitting.

Comparison with Existing Models: The proposed model is compared to existing models from the literature on the remaining 30% of the testing data for both datasets. We ensure a fair comparison by attempting to find the best hyperparameter settings for the existing models that minimize their errors.

Results: The proposed model outperforms existing models across all six evaluation criteria (MSE, bias, variance, AIC, Adjusted R-squared) for both datasets as outlined in Table 1 and Table 3. The predicted detected and corrected faults at the end of testing time is listed in Table 2 and 4. The predicted faults from the testing data are plotted against the actual values, demonstrating a better prediction compared to the existing models in Fig. 1 and Fig. 2. While the existing models require a higher number of training epochs to achieve convergence, they still exhibit higher loss values compared to the proposed model, even after hyperparameter tuning.

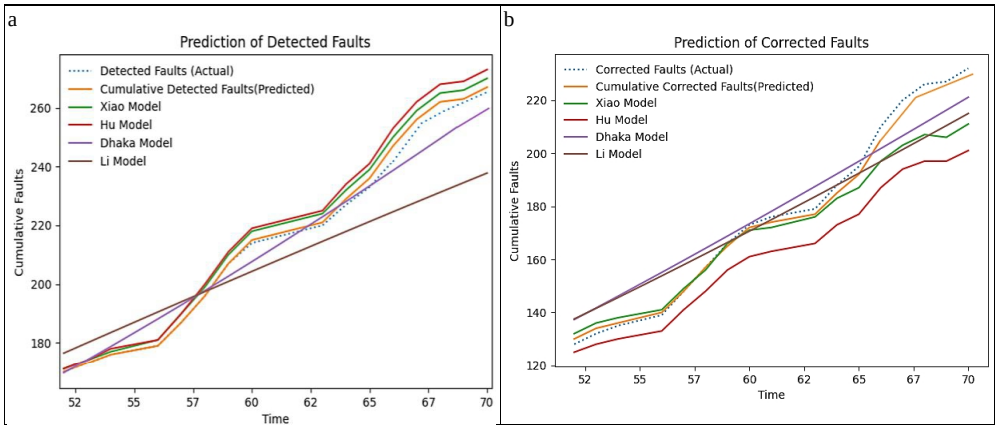

Fig. 1. Predicted faults of Dataset 1.

Table 1. Model comparison of Dataset 1.

Metric	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model
MSE	13.176	34.588	59.471	12.910	130.371	49.763	314.471	149.765	55.297	535.420
Adjusted R-squared	1.000	1.000	1.000	0.999	0.994	0.999	0.998	1.000	0.996	0.959
Bias	-2.353	-4.941	-6.412	-0.177	-1.033	4.824	15.059	10.941	1.885	-6.280
Variation	2.849	3.288	4.417	3.621	11.474	5.423	9.653	5.651	7.258	22.472
AIC	80.260	85.607	90.646	304.167	433.587	90.839	116.446	93.388	385.633	512.674

Table 2. End point prediction of faults.

Models	Actual	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model
Detected Faults	265	267	271	280	256	232
Corrected Faults	232	229	215	203	223	217

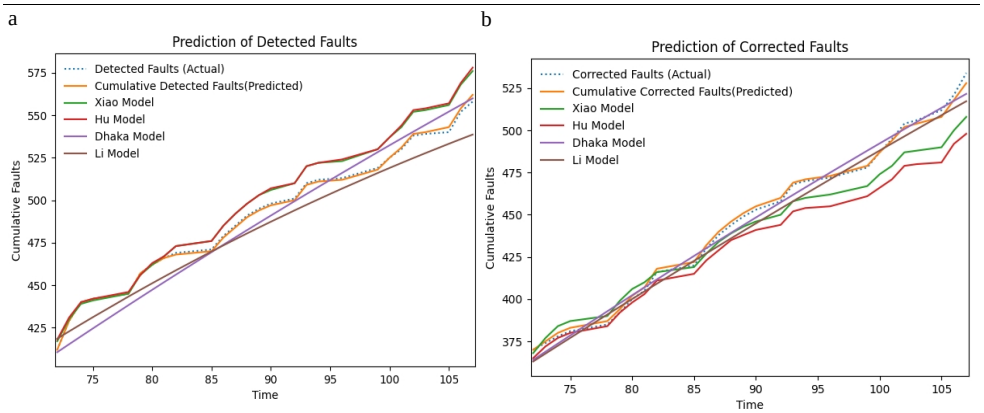

Fig. 2. Predicted faults of Dataset 2.

Table 3. Model Comparison of Dataset 2.

Metric	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model
MSE	4.148	39.333	38.407	90.557	159.452	4.852	128.889	269.185	111.071	95.147
Adjusted R-squared	1.000	1.000	1.000	0.999	0.999	1.000	1.000	1.000	0.999	0.999
Bias	0.296	-4.222	-5.444	-0.226	-1.343	-0.556	6.370	12.741	-1.275	-0.516
Variation	2.053	4.726	3.017	9.567	12.627	2.172	9.576	10.534	10.521	9.796
AIC	96.888	157.243	128.731	655.603	705.592	72.393	196.225	198.678	673.776	659.059

Table 4. End point fault prediction of Dataset 2.

Models	Actual	Proposed Model	Xiao Model	Hu Model	Dhaka Model	Li Model
Detected Faults	556	558	575	578	557	533
Corrected Faults	526	523	508	499	518	513

4.4.2. Severity Prediction

In addition to predicting the overall number of detected and corrected faults, our proposed Deep Neural Network (DNN)-based Software Reliability Growth Model (SRGM) uniquely incorporates the prediction of fault severity. The model classifies corrected faults into minor, major, and critical categories. To the best of our knowledge, this is the first application of an Artificial Neural Network (ANN) approach to fault severity prediction. Traditional parametric severity models often rely on unrealistic assumptions, such as instantaneous fault correction, and involve numerous parameters, leading to potential overfitting. In contrast, our non-parametric DNN-based model effectively addresses these limitations. We evaluated the model's performance on the remaining 30% of the testing data for both datasets. Results, presented in a Table 5, demonstrate that the predicted severity distribution closely aligns with the actual severity distribution of corrected faults, with plots (Fig. 3) visually demonstrating the model's accuracy compared to actual values. We evaluated the predicted results using error metrics discussed in Table 5, showcasing that our proposed model predicts fault severity with less error.

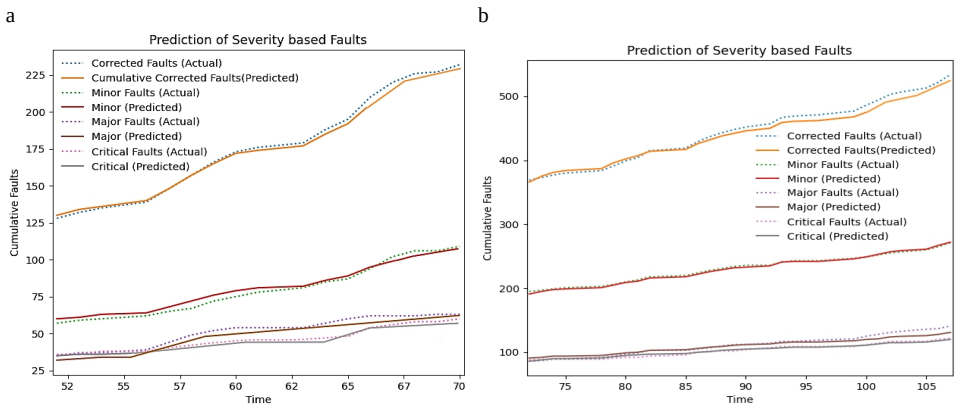

Fig. 3. Severity based fault prediction of dataset 1(a) and Dataset 2(b).

Table 5. Error Metrics for severity prediction.

Error	Dataset 1			Dataset 2		
	Minor	Major	Critical	Minor	Major	Critical
MSE	9.8235	25.705	14.235	3.222	22.629	3.2222
Adjusted R-squared	0.998	0.997	0.990	0.999	0.998	0.999
Bias	1.117	-9.823	-4.588	-0.925	-1.370	0.111
Variation	3.0183	3.7456	4.9631	1.5671	4.6422	1.8257
AIC	88.527	74.463	67.495	64.066	57.586	50.213

Table 6. Step-wise fault prediction for dataset 1 and 2.

Time	Actual Minor Faults	Predicted Minor Faults	Actual Major Faults	Predicted Major Faults	Actual Critical Faults	Predicted Critical Faults	Total Corrected Faults	Predicted Corrected Faults	Time	Actual Minor Faults	Predicted Minor Faults	Actual Major Faults	Predicted Major Faults	Actual Critical Faults	Predicted Critical Faults	Total Corrected Faults	Predicted Corrected Faults
52	57	60	35	32	36	35	128	127	72	195	191	86	91	88	86	369	368
53	59	61	37	34	36	36	132	131	73	197	195	87	92	89	88	373	375
54	60	63	37	36	38	36	135	135	74	199	198	89	94	89	90	377	382
55	60	63	37	36	38	36	135	135	75	201	199	90	94	89	90	380	383
56	62	64	39	37	38	37	139	138	76	201	199	90	94	89	90	380	383
57	65	68	44	38	39	38	148	144	77	201	199	90	94	89	90	380	383
58	67	72	49	39	41	40	157	151	78	203	201	92	95	89	91	384	387
59	72	76	52	41	42	41	166	158	79	206	205	94	97	91	93	391	395
60	75	79	54	50	44	42	173	171	80	210	209	97	99	92	95	399	403
61	78	81	54	52	44	42	176	175	81	213	211	99	100	92	96	404	407
62	78	81	54	52	44	42	176	175	82	218	216	103	103	94	97	415	416
63	81	82	54	54	44	43	179	179	83	218	216	103	103	94	97	415	416
64	85	84	57	54	46	44	188	182	84	218	216	103	103	94	97	415	416
65	87	86	60	56	48	49	195	191	85	220	218	103	104	96	98	419	420
66	94	88	62	58	54	53	210	199	86	224	222	105	106	100	100	429	428
67	102	99	62	59	56	55	220	213	87	228	226	107	108	102	101	437	435
68	106	104	62	60	58	56	226	220	88	231	229	110	109	102	103	443	441
69	106	107	63	62	58	57	227	226	89	234	232	112	111	102	104	448	447
70	109	108	63	63	60	58	232	229	90	236	233	112	112	104	105	452	450
									91	236	233	112	112	104	105	452	450
									92	236	235	114	113	107	106	457	454
									93	241	241	115	115	109	107	465	463
									94	243	242	117	116	109	108	469	466
									95	243	243	117	116	109	108	469	467
									96	243	243	119	116	109	108	471	467
									97	243	244	119	217	109	108	471	569
									98	243	244	119	217	110	109	472	570
									99	247	246	121	118	110	110	478	474
									100	249	249	125	121	112	111	486	481
									101	252	253	128	123	114	113	494	489
									102	255	257	131	126	117	115	503	498
									103	257	259	133	127	117	116	507	502
									104	257	260	133	127	117	116	507	503
									105	260	261	136	129	117	117	513	507
									106	265	267	137	131	120	119	522	517
									107	271	270	141	135	122	121	534	526

The extensive evaluation on two datasets demonstrates that the proposed DNN-based SRGM offers a significant improvement over existing models. Its ability to leverage BiGRU layers, effective hyperparameter selection, and early stopping mechanism contribute to its superior performance in accurately predicting software testing outcomes, including both **detected and corrected faults**, while also considering **fault severity**. This ultimately leads to more **reliable and informative guidance** for software development teams.

5. Conclusion

This paper presented a DNN-based SRGM for predicting software testing outcomes. The key advantage of our proposed model lies in its ability to effectively handle both **fault detection and correction along with severity considerations**. While other models lack the ability to address these crucial aspects of software testing data. The proposed model stands out by predicting the severity (minor, major, critical) of corrected faults using an DNN approach, which is novel in the current literature. This capability provides valuable insights for prioritizing resources and addressing critical issues proactively. Through extensive evaluation and comparison with existing parametric and non-parametric models, the proposed model consistently performs well across six evaluation criteria (MSE, bias, variance, AIC, Adjusted R-squared) for both datasets, showcasing its superior ability to capture complex relationships within the data. Our non-parametric model overcomes the limitations of existing parametric models, such as unrealistic assumptions and overfitting caused by a large number of parameters. As software reliability remains a critical aspect in the ever-evolving technology landscape, our DNN-based SRGM stands as a valuable tool for decision-makers, offering insights into testing efficacy, fault correction, and severity impact.

In future work, we aim to further refine and extend our model, explore additional datasets, and delve deeper into the optimization of hyperparameters for even more accurate predictions. Overall, the proposed DNN-based SRGM signifies a promising advancement in software reliability modeling, combining the strengths of neural network architectures with a holistic consideration of fault dynamics and severity predictions.

Appendix A

Table 7. Dataset 1- Tomcat Maven.

Time	No of Detected faults	Cumulative Detected fault	No of Corrected faults	Cumulative Corrected	Minor	Major	Critical	Time	No of Detected faults	Cumulative Detected fault	No of Corrected faults	Cumulative Corrected faults	Minor	Major	Critical
1	1	1	0	0	0	0	0	36	3	98	3	74	2	1	0
2	3	4	0	0	0	0	0	37	8	106	4	78	3	1	0
3	8	12	0	0	0	0	0	38	4	110	4	82	4	0	0
4	9	21	2	2	1	1	0	39	3	113	5	87	0	2	3
5	4	25	4	6	2	1	1	40	4	117	5	92	4	0	1
6	0	25	0	6	0	0	0	41	1	118	0	92	0	0	0
7	2	27	0	6	0	0	0	42	6	124	3	95	2	1	0
8	5	32	3	9	1	1	1	43	7	131	6	101	3	3	0
9	4	36	3	12	2	0	1	44	2	133	1	102	1	0	0
10	1	37	1	13	0	0	1	45	9	142	5	107	1	3	1
11	3	40	2	15	1	0	1	46	4	146	4	111	1	2	1
12	3	43	2	17	0	1	1	47	2	148	2	113	2	0	0
13	0	43	0	17	0	0	0	48	3	151	0	113	0	0	0
14	3	46	0	17	0	0	0	49	3	154	1	114	0	0	1
15	5	51	3	20	0	1	2	50	6	160	5	119	1	3	1
16	3	54	5	25	1	2	2	51	7	167	6	125	2	2	2
17	3	57	2	27	2	0	0	52	4	171	3	128	1	1	1
18	3	60	3	30	0	2	1	53	3	174	4	132	2	2	0
19	5	65	5	35	1	2	2	54	3	177	3	135	1	0	2
20	0	65	0	35	0	0	0	55	0	177	0	135	0	0	0
21	1	66	0	35	0	0	0	56	3	180	4	139	2	2	0
22	4	70	3	38	2	1	0	57	8	188	9	148	3	5	1
23	1	71	2	40	0	0	2	58	9	197	9	157	2	5	2
24	2	73	3	43	2	1	0	59	5	202	9	166	5	3	1
25	5	78	4	47	2	0	2	60	7	209	7	173	3	2	2
26	2	80	3	50	0	0	3	61	2	211	3	176	3	0	0
27	1	81	0	50	0	0	0	62	4	215	0	176	0	0	0
28	0	81	0	50	0	0	0	63	4	219	3	179	3	0	0
29	2	83	3	53	2	1	0	64	7	226	9	188	4	3	2
30	3	86	6	59	2	2	2	65	6	232	7	195	2	3	2
31	2	88	5	64	3	0	2	66	9	241	15	210	7	2	6
32	6	94	5	69	4	0	1	67	8	249	10	220	8	0	2
33	1	95	2	71	2	0	0	68	5	254	6	226	4	0	2
34	0	95	0	71	0	0	0	69	2	256	1	227	0	1	0
35	0	95	0	71	0	0	0	70	4	260	5	232	3	0	2

Table 8. Dataset 2- Apache Age.

Time	No of Detected faults	Cumulative Detected fault	No of Corrected faults	Cumulative Corrected faults	Minor	Major	Critical	Time	No of Detected faults	Cumulative Detected fault	No of Corrected faults	Cumulative Corrected faults	Minor	Major	Critical
1	4	4	0	0	0	0	0	55	0	312	0	268	0	0	0
2	16	20	4	4	2	0	2	56	1	313	4	272	2	2	0
3	5	25	1	5	0	1	0	57	8	321	5	277	3	0	2
4	3	28	3	8	2	1	0	58	7	328	5	282	4	0	1
5	4	32	4	12	0	1	3	59	10	338	7	289	5	2	0
6	1	33	1	13	1	0	0	60	15	353	12	301	6	4	2
7	5	38	6	19	3	3	0	61	1	354	3	304	3	0	0
8	3	41	2	21	2	0	0	62	0	354	0	304	0	0	0
9	8	49	5	26	2	1	2	63	1	355	6	310	6	0	0
10	8	57	8	34	3	2	3	64	3	358	7	317	2	3	2
11	12	69	7	41	4	1	2	65	11	369	9	326	8	0	1
12	5	74	5	46	2	0	3	66	7	376	8	334	5	2	1
13	3	77	1	47	1	0	0	67	9	385	7	341	2	2	3
14	6	83	5	52	2	0	3	68	0	385	0	341		0	0
15	9	92	9	61	5	2	2	69	5	390	7	348	2	3	2
16	13	105	12	73	7	3	2	70	4	394	3	351	3	0	0
17	9	114	8	81	6	2	0	71	7	401	11	362	6	2	3
18	5	119	5	86	0	2	3	72	11	412	7	369	3	1	3
19	7	126	6	92	4	2	0	73	15	427	4	373	2	1	1

20	1	127	0	92	0	0	0	74	11	438	4	377	2	2	0
21	1	128	1	93	0	0	1	75	2	440	3	380	2	1	0
22	14	142	15	108	8	3	4	76	0	440	0	380	0	0	0
23	4	146	7	115	4	3	0	77	0	440	0	380	0	0	0
24	9	155	9	124	5	2	2	78	3	443	4	384	2	2	0
25	12	167	11	135	4	3	4	79	12	455	7	391	3	2	2
26	10	177	4	139	4	0	0	80	5	460	8	399	4	3	1
27	0	177	0	139	0	0	0	81	4	464	5	404	3	2	0
28	4	181	6	145	6	0	0	82	3	467	11	415	5	4	2
29	9	190	11	156	5	2	4	83	0	467	0	415	0	0	0
30	14	204	12	168	6	3	3	84	0	467	0	415	0	0	0
31	8	212	8	176	3	3	2	85	2	469	4	419	2	0	2
32	6	218	7	183	7	0	0	86	8	477	10	429	4	2	4
33	3	221	3	186	0	2	1	87	6	483	8	437	4	2	2
34	0	221	0	186	0	0	0	88	6	489	6	443	3	3	0
35	5	226	7	193	4	0	3	89	4	493	5	448	3	2	0
36	9	235	13	206	6	3	4	90	3	496	4	452	2	0	2
37	6	241	9	215	4	2	3	91	0	496	0	452	0	0	0
38	9	250	3	218	2	1	0	92	3	499	5	457	0	2	3
39	3	253	3	221	0	2	1	93	9	508	10	467	5	3	2
40	0	253	0	221	0	0	0	94	2	510	2	469	2	0	0
41	2	255	1	222	0	1	0	95	0	510	0	469	0	0	0
42	0	255	0	222	0	0	0	96	1	511	2	471	0	2	0
43	2	257	3	225	0	3	0	97	0	511	0	471	0	0	0
44	3	260	3	228	0	2	1	98	0	511	0	471	0	0	0
45	8	268	6	234	4	2	0	99	6	517	6	477	4	2	0
46	6	274	3	237	2	0	1	100	6	523	9	486	2	4	3
47	10	284	5	242	1	2	2	101	5	528	8	494	3	3	2
48	0	284	0	242	0	0	0	102	8	536	9	503	3	3	3
49	1	285	3	245	3	0	0	103	1	537	4	507	2	2	0
50	5	290	4	249	2	2	0	104	0	537	0	507	0	0	0
51	3	293	6	255	3	1	2	105	1	538	6	513	3	3	0
52	5	298	6	261	2	2	2	106	12	550	9	522	5	1	3
53	5	303	3	264	1	0	2	107	6	556	12	534	6	4	2
54	9	312	4	268	3	0	1								

References

Bisi, M., Goyal, N. K. 2016. Software development efforts prediction using artificial neural network. *IET Software*, *10*(3), 63–71. https://doi.org/10.1049/IET-SEN.2015.0061

Dhaka, V., Nijhawan, N. 2022. Effect of change in environment on reliability growth modeling integrating fault reduction factor and change point: a general approach. *Annals of Operations Research*. https://doi.org/10.1007/s10479-022-05084-6

Hsu, C. J., Huang, C. Y., Chang, J. R. 2011. Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor. *Applied Mathematical Modelling*, *35*(1), 506–521. https://doi.org/10.1016/J.APM.2010.07.017

Hu, Q. P., Xie, M., Ng, S. H., Levitin, G. 2007. Robust recurrent neural network modeling for software fault detection and correction prediction. *Reliability Engineering & System Safety*, *92*(3), 332–340. https://doi.org/10.1016/J.RESS.2006.04.007

Huang, Y. S., Chiu, K. C., Chen, W. M. 2022. A software reliability growth model for imperfect debugging. *Journal of Systems and Software*, *188*, 111267. https://doi.org/10.1016/j.jss.2022.111267

Karunanithi, N., Malaiya, Y. K., Whitley, D. 1991. Prediction of software reliability using neural networks. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 124–130. https://doi.org/10.1109/ISSRE.1991.145366

Li, C., Zheng, J., Okamura, H., Dohi, T. 2022. Software Reliability Prediction through Encoder-Decoder Recurrent Neural Networks. *International Journal of Mathematical, Engineering and Management Sciences*, *7*(3), 325–340. https://doi.org/10.33889/IJMEMS.2022.7.3.022

Li, Q., Pham, H. 2017. NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. *Applied Mathematical Modelling*, *51*, 68–85. https://doi.org/10.1016/j.apm.2017.06.034

Pandey, A. K., Goyal, N. K. 2010. Fault Prediction Model by Fuzzy Profile Development of Reliability Relevant Software Metrics. *International Journal of Computer Applications*, *11*(6), 34–41. https://doi.org/10.5120/1584-2124

Peng, R., Li, Y. F., Liu, Y. 2018. TEF dependent software FDP and FCP models. *SpringerBriefs in Computer Science*, 15–32. https://doi.org/10.1007/978-981-13-1162-8_3/TABLES/1

Pradhan, V., Kumar, A., Dhar, J. 2022. Modelling software reliability growth through generalized inflection S-shaped fault reduction factor and optimal release time. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *236*(1), 18–36. https://doi.org/10.1177/1748006X211033713/ASSET/IMAGES/LARGE/10.1177_1748006X211033713-FIG5.JPEG

Singhal, S., Kapur, P. K., Kumar, V., Panwar, S. 2023. Stochastic debugging based reliability growth models for Open Source Software project. *Annals of Operations Research*, 1–39. https://doi.org/10.1007/S10479-023-05240-6/TABLES/9

Wang, J., Zhang, C. 2018. Software reliability prediction using a deep learning model based on the RNN encoder–decoder. *Reliability Engineering and System Safety*, *170*, 73–82. https://doi.org/10.1016/J.RESS.2017.10.019

Xiao, H., Cao, M., Peng, R. 2020. Artificial neural network based software fault detection and correction prediction models considering testing effort. *Applied Soft Computing*, *94*, 106491. https://doi.org/10.1016/J.ASOC.2020.106491

Yang, J., Liu, Y., Xie, M., Zhao, M. 2016. Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes. *Journal of Systems and Software*, *115*, 102–110. https://doi.org/10.1016/j.jss.2016.01.025