

# Fault Big Data Analysis Tool based on Deep Learning

Yoshinobu Tamura<sup>a,\*</sup> and Shigeru Yamada<sup>b</sup>

<sup>a</sup>*Tokyo City University, Tamazutsumi 1-28-1, Setagaya-ku, Tokyo, 158-8557, Japan*

<sup>b</sup>*Tottori University, Minami 4-101, Koyama, Tottori-shi, 680-8552, Japan*

---

## Abstract

Software managers can obtain useful information from many fault data sets recorded on bug tracking systems (BTS). However, it is difficult to find helpful measures for software reliability, maintainability, and performability, because the data collected on the BTS are mixed with qualitative and quantitative ones. This paper discusses the methods of reliability, maintainability, and performability assessment by deep learning for big data in terms of software faults. Specifically, we implement the reliability, maintainability, and performability analysis tool discussed in our method by using the latest programming technology. Moreover, we show several performance examples of the implemented application software by using the fault big data observed in the practical projects.

**Keywords:** open source software; software tool; reliability; maintainability; performability; fault big data; deep learning

(Submitted on October 15, 2017; Revised on February 28, 2018; Accepted on March 30, 2018)

© 2019 Totem Publisher, Inc. All rights reserved.

---

## 1. Introduction

In this paper, we propose the fault big data analysis tool implemented useful assessment measures such as performability. Various software reliability grows models [1-2] have been developed in order to manage and assess the quality under the system testing in software development and management. However, it is difficult to assess the reliability of open source software (OSS), because the development and maintenance cycle of OSS is different from the conventional software development styles. In the past, several reliability models focused on OSS have been developed and proposed [3].

Bug tracking systems (BTS) are used in many OSS projects from the characteristics of OSS development paradigm. In the BTS, fault data are collected and recorded by the computer software used by OSS managers, developers, and users. It will be useful for OSS managers, developers, and users to assess the performability, maintainability, and reliability of OSS, if all the fault data collected on the BTS are effectively used for software quality and reliability improvement. Then, OSS reliability, maintainability, and performability can be assessed based on the fault data by using an AI approach such as machine learning. This paper focuses on the fault big data analysis tool based on deep learning by OSS fault big data. Then, we discuss the method of OSS performability, maintainability, and reliability assessment by using deep learning. In particular, we implement the application software for reliability, maintainability, and performability analysis through the AI approach. Then, the implemented application software is based on the latest implementation technologies such as NW.js, HTML, JavaScript, CSS3, and statistical computing R programming language including ggplot2 and plotly packages. In addition, several performance examples of our implemented tool based on the fault big data of practical OSS project are shown.

## 2. Assessment Measures of OSS based on Deep Learning

In the past, several deep learning algorithms have been proposed [4-6]. This paper focuses on the deep neural network to learn the fault big data of OSS projects. Then, the coefficient value considering the software reliability trend is used as the objective variable. Table 1 shows the relationship of coefficient values for reliability trends.

---

\* Corresponding author.

E-mail address: [tamuray@tcu.ac.jp](mailto:tamuray@tcu.ac.jp)

Table 1. The coefficient values

Reliability trends	Coefficient value
Increase	Index: 1
Decrease	Index: -1

Two kinds of reliability trend levels are used as the coefficient values: Increase and Decrease. The factors of pre-training units, such as the following items, are selected as nine kinds of explanatory variables.

- Date
- Product Name
- Component Name
- Software Version
- Reporter Nickname
- Assignee Nickname
- Fault Status
- OS
- Level of Fault

Then, several explanatory variables, such as character data, are converted to numerical values.

The mean time between software failures (MTBF) is helpful to understand the property of software failure-occurrence. We define the MTBF as follows:

$$t_k = t_{k-1} + dl_{k-1} \cdot SD_{k-1} \quad (1)$$

Where  $t_k$  is the  $k^{\text{th}}$  MTBF and  $dl_k$  is the  $k^{\text{th}}$  index number as the coefficient value based on deep learning, e.g., the coefficient value "1" means the growth trend in terms of reliability, while the coefficient value "-1" represents the regression trend. Also,  $SD_k$  is the  $k^{\text{th}}$  standard deviation obtained as follows:

$$SD_k = \sqrt{\frac{1}{k-1} \sum_{n=1}^k \left( t_n - \frac{1}{n} \sum_{l=1}^n t_l \right)^2} \quad (2)$$

Similarly, the correction time between software failures (CTBF) is helpful to comprehend the frequency of software maintenance. In the case of CTBF, the ten kinds of explanatory variables include "Changed date", and the factors of explanatory variables are set to the amount of training units.

Then, we define the time between software fault corrections (MTBC) as follows:

$$c_k = c_{k-1} + dl_{k-1} \cdot SD_{k-1}, \quad (3)$$

Where  $t_k$  is the  $k^{\text{th}}$  CTBF and  $dl_k$  is the estimated  $k^{\text{th}}$  index number as the coefficient value based on the correction time by using deep learning. In addition,  $SD_k$  is the following  $k^{\text{th}}$  standard deviation:

$$SD_k = \sqrt{\frac{1}{k-1} \sum_{n=1}^k \left( c_n - \frac{1}{n} \sum_{l=1}^n c_l \right)^2} \quad (4)$$

Moreover, the performability is helpful for OSS managers to understand the properties of the conditions of the software execution environment. We define the performability of OSS as follows:

$$p_k = \frac{t_k}{t_k + c_k} \quad (5)$$

Where  $p_k$  is the  $k^{\text{th}}$  performability.  $p_k$  is decided by using the estimated MTBF and MTBC. Therefore, we can obtain the following performability of OSS from Equations (2) and (4):

$$p_k = \frac{t_{k-1} + dl_{k-1} \cdot SD_{k-1}}{t_{k-1} + dl_{k-1} \cdot SD_{k-1} + c_{k-1} + dl_{k-1} \cdot SD_{k-1}} \quad (6)$$

Finally, the performability of OSS is given by the following equation:

$$p_k = \frac{t_{k-1} + dl_{k-1} \cdot \frac{1}{k-2} \sum_{n=1}^{k-1} \left( t_n - \frac{1}{n} \sum_{l=1}^n t_l \right)^2}{t_{k-1} + dl_{k-1} \cdot \sqrt{\frac{1}{k-2} \sum_{n=1}^{k-1} \left( t_n - \frac{1}{n} \sum_{l=1}^n t_l \right)^2} + c_{k-1} + dl_{k-1} \cdot \sqrt{\frac{1}{k-1} \sum_{n=1}^k \left( c_n - \frac{1}{n} \sum_{l=1}^n c_l \right)^2}} \quad (7)$$

### 3. Implementation of Software Tool

Our tool is based on latest implementation technologies such as NW.js [7], HTML, JavaScript, CSS3, and statistical computing R programming languages [8] including ggplot2 and plotly packages [9]. In particular, our software application is based on the structure shown in Figure 1 [10-11]. The simplified source code of implemented application software is coded as follows:

---

----- Partial source code of index.html -----

---

```
<html>
<meta charset="utf-8">
<body onload="init();">
<script type="text/javascript">
  function init() {
    var script = document.createElement('script');
    script.src = 'main.js';
    document.body.appendChild(script);
  }
  function mtbfc() {
    var script = document.createElement('script');
    script.src = 'mtbfc.js';
    document.body.appendChild(script);
  }
  function mtbcc() {
    var script = document.createElement('script');
    script.src = 'mtbcc.js';
    document.body.appendChild(script);
  }
  function mtbf() {
    window.open('data/mtbf.html');
  }
  function mtbc() {
    window.open('data/mtbc.html');
  }
  function mamtbfc() {
    window.open('data/mamtbfc.html');
  }
  function mamtbc() {
    window.open('data/mamtbc.html');
  }
  function lmamtbfc() {
    window.open('data/lmamtbfc.html');
  }
  function lmamtbc() {
    window.open('data/lmamtbc.html');
  }
  function per() {
    window.open('data/per.html');
  }
  function lper() {
    window.open('data/lper.html');
  }
</script>
</body>
```

---

</html>

```
mtbfc.js:
  exec = require('child_process').exec;
  exec('R --slave --file=r/mtbf.r');
```

```
mtbcc.js:
  exec = require('child_process').exec;
  exec('R --slave --file=r/mtbc.r');
```

----- Partial source code of ggplot2 and plotly -----

```
library(ggplot2)
library(plotly)

mtbf <- ggplotly(mtbfc)
htmlwidgets::saveWidget(mtbfc, "/Applications/FBDAT Based on DL.app/Contents/Resources/app.nw/data/mtbf.html", selfcontained = FALSE)
dev.off()

mamtbfc <- ggplotly(mamtbfc)
htmlwidgets::saveWidget(mamtbfc, "/Applications/FBDAT Based on DL.app/Contents/Resources/app.nw/data/mamtbfc.html", selfcontained = FALSE)
dev.off()

lmamtbfc <- ggplotly(lmamtbfc)
htmlwidgets::saveWidget(lmamtbfc, "/Applications/FBDAT Based on DL.app/Contents/Resources/app.nw/data/lmamtbfc.html", selfcontained = FALSE)
dev.off()

per <- (lmamtbfc)/(lmamtbfc+lmaintenance)
lper <- (lmamtbfc)/(lmamtbfc+lmaintenance)
dper <- data.frame(x = lmx, y = per, DATA = "Deep Learning")

maper <- ggplotly(maper)
htmlwidgets::saveWidget(maper, "/Applications/FBDAT Based on DL.app/Contents/Resources/app.nw/data/per.html", selfcontained = FALSE)
dev.off()

lmaper <- ggplotly(lmaper)
htmlwidgets::saveWidget(lmaper, "/Applications/FBDAT Based on DL.app/Contents/Resources/app.nw/data/lper.html", selfcontained = FALSE)
dev.off()
```

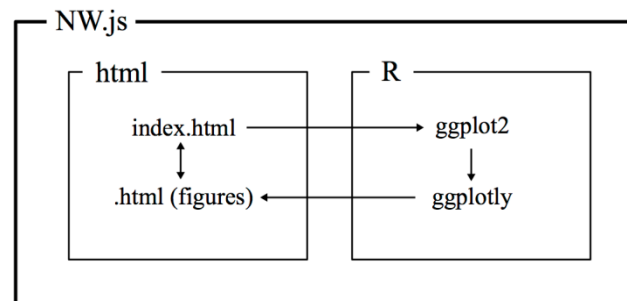


Figure 1. The framework of deep learning in our tool

This application software can be easily implemented by using the ggplot2 and plotly in R library such as the above source code.

#### 4. Performance Illustrations of Developed Software Tool

In this paper, we use the data of Apache HTTP server [12] to evaluate the performance and illustrations of the developed tool.

First, the partial data in terms of MTBF used in this paper is shown in Table 1. "Opened" to "severity" in the header in Table 1 represent the explanatory variables. The last header shows the objective variables. The numerical values in terms of each explanatory variable shown in Table 1 are converted to numerical values.

Similarly, the partial data in terms of MTBC used in this paper are shown in Table 2. From Tables 1 and 2, we can estimate the performability from these data in terms of MTBF and MTBC. Then, all data sets include 10,000 lines.

Table 1. The partial data in terms of MTBF

opened	product	component	version	reporter	assignee	status	os	severity	mtbf
2.33883102	0.051	0.024	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0.05108796	0.051	0.0346	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0.15591435	0.051	0.0269	0.0128	0.0003	0.4981	0.7856	0.38025484	0.4664	high
0.7933912	0.051	0.024	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0.04652778	0.051	0.0346	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	low
0.53811343	0.051	0.0017	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.56766204	0.051	0.024	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.54798611	0.051	0.0269	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	low
0.25594907	0.051	0.0135	0.005	0.0114	0.4981	0.7856	0.38025484	0.4664	low
1.27369213	0.051	0.024	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.91871528	0.051	0.0346	0.0042	0.0002	0.4981	0.7856	0.38025484	0.4664	low
1.63028935	0.051	0.0135	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.88961806	0.051	0.0135	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
1.90168982	0.051	0.0346	0.0277	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0.20707176	0.051	0.0346	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0.48886574	0.0899	0.0367	0.0215	0.0002	0.0923	0.0809	0.38025484	0.4664	high
0.00168982	0.051	0.024	0.0277	0.0002	0.4981	0.7856	0.38025484	0.4664	low
0.27810185	0.051	0.0017	0.0024	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.15607639	0.051	0.0135	0.0277	0.0027	0.4981	0.7856	0.38025484	0.4664	low
0.05952546	0.051	0.0269	0.0128	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0.21253472	0.051	0.024	0.0128	0.0114	0.4981	0.7856	0.38025484	0.4664	high
2.62094907	0.051	0.0269	0.0128	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.08758102	0.0245	0.1147	0.0045	0.0003	0.4981	0.7856	0.38025484	0.4664	low
0.00628472	0.051	0.0135	0.0277	0.0003	0.4981	0.7856	0.38025484	0.4664	low
0.00987269	0.051	0.0269	0.0128	0.0003	0.4981	0.7856	0.38025484	0.4664	high
0.99197917	0.051	0.0346	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.30546296	0.051	0.0135	0.0277	0.0114	0.4981	0.7856	0.38025484	0.4664	low
1.22015046	0.051	0.0135	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0.32399306	0.051	0.0037	0.005	0.0003	0.4981	0.7856	0.38025484	0.4664	low
0.05876157	0.051	0.0346	0.0277	0.0006	0.4981	0.7856	0.38025484	0.4664	low
0.1780787	0.051	0.0269	0.0024	0.0114	0.4981	0.7856	0.38025484	0.4664	high

Table 2. The partial data in terms of MTBC

changed	product	component	version	reporter	assignee	status	os	severity	mtbc
1124.04796	0.051	0.024	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0346	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
271.703681	0.051	0.0269	0.0128	0.0003	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.024	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	low
1357.03872	0.051	0.0346	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0017	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0	0.051	0.024	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	low
6.14890046	0.051	0.0269	0.0042	0.0114	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0135	0.005	0.0114	0.4981	0.7856	0.38025484	0.4664	low
1287.6775	0.051	0.024	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0346	0.0042	0.0002	0.4981	0.7856	0.38025484	0.4664	low
1353.07434	0.051	0.0135	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0135	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0	0.051	0.0346	0.0277	0.0114	0.4981	0.7856	0.38025484	0.4664	low
1166.17413	0.051	0.0346	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	high
1338.15117	0.0899	0.0367	0.0215	0.0002	0.0923	0.0809	0.38025484	0.4664	high
0	0.051	0.024	0.0277	0.0002	0.4981	0.7856	0.38025484	0.4664	low
579.496539	0.051	0.0017	0.0024	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0135	0.0277	0.0027	0.4981	0.7856	0.38025484	0.4664	low
1309.52204	0.051	0.0269	0.0128	0.0001	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.024	0.0128	0.0114	0.4981	0.7856	0.38025484	0.4664	low
0	0.051	0.0269	0.0128	0.0001	0.4981	0.7856	0.38025484	0.4664	low
1724.93087	0.0245	0.1147	0.0045	0.0003	0.4981	0.7856	0.38025484	0.4664	high
0	0.051	0.0135	0.0277	0.0003	0.4981	0.7856	0.38025484	0.4664	low
802.913044	0.051	0.0269	0.0128	0.0003	0.4981	0.7856	0.38025484	0.4664	high
640.951806	0.051	0.0346	0.0042	0.0001	0.4981	0.7856	0.38025484	0.4664	low
0	0.051	0.0135	0.0277	0.0114	0.4981	0.7856	0.38025484	0.4664	low
0	0.051	0.0135	0.0277	0.0001	0.4981	0.7856	0.38025484	0.4664	low
348.356748	0.051	0.0037	0.005	0.0003	0.4981	0.7856	0.38025484	0.4664	high
136.232049	0.051	0.0346	0.0277	0.0006	0.4981	0.7856	0.38025484	0.4664	low
1382.28894	0.051	0.0269	0.0024	0.0114	0.4981	0.7856	0.38025484	0.4664	high

Second, we show the main screen, main menu, and readme screens of our software application in Figures 2 and 3, respectively. The estimated 10-fault moving average MTBF by using deep learning based on testing data is recorded from the BTS in Figure 4. Similarly, the estimated 10-fault moving average MTBC based on deep learning by testing data is obtained from the BTS in Figure 5.

Moreover, Figure 6 shows the estimated 50-fault moving average MTBF by deep learning based on the testing data sets. Similarly, Figure 7 shows the estimated 50-fault moving average MTBC by deep learning based on the testing data sets. In particular, we found that our tool will be helpful for OSS managers and developers to understand the maintainability and reliability trends in the standpoint of the long-term prediction from Figures 6 and 7.

Furthermore, Figure 8 shows the estimated 10-fault moving average performability by deep learning based on the testing data sets. Similarly, Figure 9 shows the estimated 50-fault moving average performability by deep learning based on the testing data sets. Figures 8 and 9 show the interesting results. In particular, from Figure 8, we found that the performability decreases as a whole. In addition, Figure 9 shows that the variation of performability becomes large as the operation proceeds. According to Figures 8 and 9, the Apache HTTP server frequently made modifications according to the version changes.

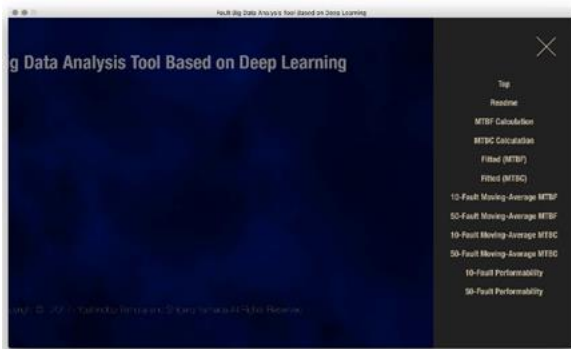


Figure 2. The main menu of our software application



Figure 3. The readme screen of our software application

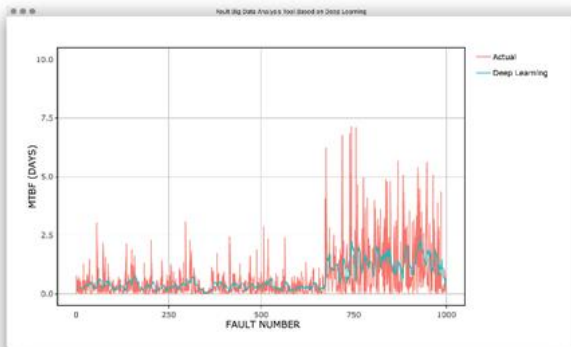


Figure 4. The estimated 10-fault moving average MTBF

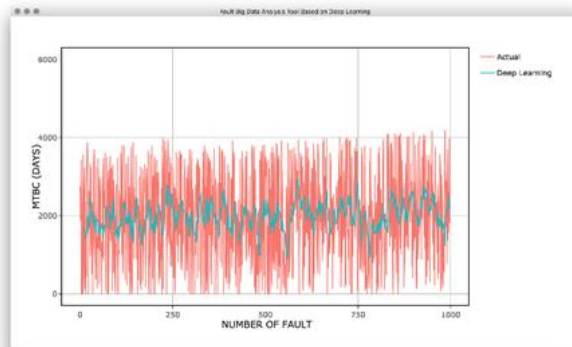


Figure 5. The estimated 10-fault moving average MTBC

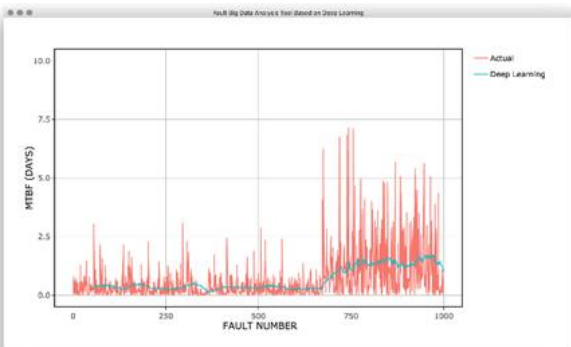


Figure 6. The estimated 50-fault moving average MTBF

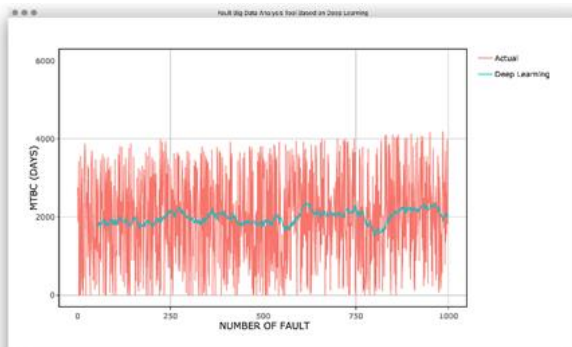


Figure 7. The estimated 50-fault moving average MTBC

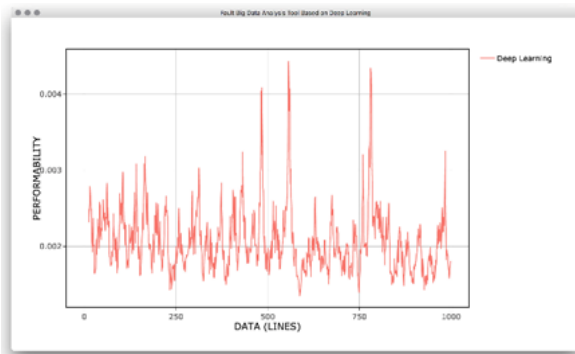


Figure 8. The estimated 10-fault moving average performability

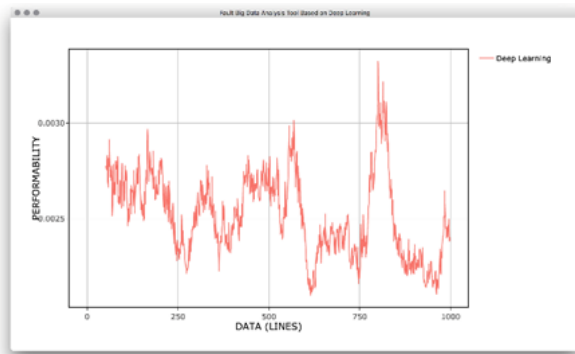


Figure 9. The estimated 50-fault moving average performability

## 5. Concluding Remarks

The BTS is the support tool for OSS maintenance and development based on the fault database. It will be helpful for OSS users, developers, and managers to confirm the OSS reliability, maintainability and performability, if all data sets uploaded on the BTS are effectively used for OSS project development and management. This paper has discussed the method of OSS dependability, maintainability, and performability assessment by using deep learning based on fault big data on BTS. The implemented application software uses the latest application development technologies such as NW.js, JavaScript, CSS3, HTML, and statistical computing R programming languages. Then, we can easily develop the application software by using NW.js and R, including ggplot2 and plotly packages. In addition, this paper has shown several performance examples of OSS reliability, maintainability, and performability assessment based on fault big data in practical OSS projects. The implemented our software is available at "<http://tam.ims.tcu.ac.jp/>".

## Acknowledgments

This work was supported in part by the JSPS KAKENHI (No. 16K01242) in Japan.

## References

1. S. Yamada, "Software Reliability Modeling: Fundamentals and Applications," Springer-Verlag, Tokyo/Heidelberg, 2014
2. P. K. Kapur, H. Pham, A. Gupta, and P.C. Jha, "Software Reliability Assessment with OR Applications," Springer-Verlag, London, 2011
3. S. Yamada and Y. Tamura, "OSS Reliability Measurement and Assessment," Springer International Publishing, Switzerland, 2016
4. D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-Supervised Learning with Deep Generative Models," in *Proceedings of Neural Information Processing Systems*, 2014
5. E. D. George, Y. Dong, D. Li, and A. Alex, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 20, No. 1, pp. 30-42, 2012
6. B. Hutchinson, L. Deng, and D. Yu, "Tensor Deep Stacking Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 8, pp. 1944-1957, 2013
7. NW.js community, NW.js, <http://nwjs.io/>
8. The R Project for Statistical Computing, The R Foundation, <https://www.r-project.org/>
9. Plotly R Library, Plotly, <https://plot.ly/r/>
10. Y. Tamura and S. Yamada, "Reliability Analysis based on Deep Learning for Fault Big Data on Bug Tracking System," in *Proceedings of the IEEE International Conference on Reliability, Infocom Technology and Optimization*, pp. 37-42, Noida, India, September 7-9, 2016
11. Y. Tamura and S. Yamada, "Reliability and Maintainability Analysis and its Tool based on Deep Learning for Fault Big Data," in *Proceedings of the IEEE International Conference on Reliability, Infocom Technology and Optimization*, pp. 104-109, Noida, India, September 20-22, 2017
12. The Apache Software Foundation, The Apache HTTP Server Project, <http://httpd.apache.org/>