Empirical Research on the Application of a Structure-Based Software Reliability Model

Jie Zhang, Yang Lu, Ke Shi, and Chong Xu

***Abstract*—Reliability engineering implemented early in the development process has a significant impact on improving software quality. It can assist in the design of architecture and guide later testing, which is beyond the scope of traditional reliability analysis methods. Structural reliability models work for this, but most of them remain tested in only simulation case studies due to lack of actual data. Here we use software metrics for reliability modeling which are collected from source codes of post versions. Through the proposed strategy, redundant metric elements are filtered out and the rest are aggregated to represent the module reliability. We further propose a framework to automatically apply the module value and calculate overall reliability by introducing formal methods. The experimental results from an actual project show that reliability analysis at the design and development stage can be close to the validity of analysis at the test stage through reasonable application of metric data. The study also demonstrates that the proposed methods have good applicability.**

***Index Terms*—Algebraic method, reliability evaluation, software metrics, software reliability.**

I. Introduction

THE essence of software reliability engineering lies in improving software quality, and its role covers all development stages. Most software reliability studies use failure data from the test stage, which focuses on predicting the relatively stable interval of the reliability growth curve and providing the basis for adjustment of test workload and software release strategy. Recent research suggests that reliability evaluation in the early stages has important implications for avoiding possible revision costs in the middle and later stages of development, especially for reliability-sensitive and safety-critical software systems [1]–[4]. Generally, in order to optimize software design, appropriate methods can be introduced as early as possible to evaluate the overall reliability of different architectures. Structural reliability modeling works for this, which takes a component as a basic granularity and considers interaction modes and structural styles of component compositions. Typical models include semi-Markov process (SMP) [5], discrete time Markov chain (DTMC) [6], and continuous-time Markov chain (CTMC) [7]. Compared with the traditional reliability models such as Goel-Okumoto (G-O) [8], they belong to a Markovian model which is based on a stochastic modeling method.

Markovian models will be state-explosion-prone when used in large-scale and complex software. As such, state-free models are also used to evaluate software system recently. Zheng *et al*. [9] present an analytical approach based on Pareto distribution for performance estimation of unreliable infrastructure-as-a-service (IaaS) clouds. Li *et al*. [10] employ an autoregressive moving average model in time series for quality-of-service (QoS) prediction of real composite services. The above method considers the dynamic change in continuous runtime, but does not focus on the static analysis of component structure. Xia *et al*. [11] use a stochastic-Petri-net to calculate the process-normal-completion probability as the reliability estimate since Petri-nets are highly capable of describing complex component-based software. However, this study still needs experimental data and is not suitable for software reliability evaluation at early design stage. In contrast, the structural model represented by DTMC is more suitable for a reliability assessment at this stage, because it usually does not require runtime data.

Stochastic modeling is limited in practical applications of structure-based software reliability analysis. Take the DTMC model as an example. Its two key parameters required for modeling, component reliability and control transfer probability among components, are given by simulated cases rather than from actual projects [12]. Consensus of previous research is that a single component can be regarded as a black-box, and the component parameters tend to be stable in continuous reuse and iteration. But reliability cannot be the inherent property of software components because of the difference in requirements and external environments between actual projects. Moreover, the state explosion caused by component-level modeling is also a difficult problem in the application of large-scale complex software. Recently, more attention was paid to service/cloud-based software reliability. Xia *et al*. [13] present a stochastic model based on a Poisson arrival process for quality evaluation of IaaS clouds. This work considers expected request completion time, rejection probability, and system overhead rate as key metrics, and can be used to help design and optimize cloud systems. Luo *et al*.

Manuscript received February 12, 2020; accepted April 2, 2020. This work was supported by the National Natural Science Foundation of China (61572167), the National Key Research and Development Program of China (2016YFC0801804), and the Natural Science Foundation for Anhui Higher Education Institutions of China (KJ2019A0482). Recommended by Associate Editor Xin Luo. *(Corresponding author: Yang Lu.)*

Citation: J. Zhang, Y. Lu, K. Shi, and C. Xu, "Empirical research on the application of a structure-based software reliability model," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 6, pp. 1153–1162, Jun. 2021.

J. Zhang is with the School of Computer and Information, Anhui Normal University, Wuhu 241003, China (e-mail: zjzj2526@163.com).

Y. Lu, K. Shi, and C. Xu are with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China (e-mail: luyang.hf@126.com; shike@mail.hfut.edu.cn; xu.chong@mail.hfut.edu.cn).

Digital Object Identifier 10.1109/JAS.2020.1003309

[14] propose a scheme to build the ensemble of matrix-factorization based QoS estimators and achieve an AdaBoost-based aggregation result. This work presents high estimation accuracy at low time cost. Furthermore, the authors incorporate second-order solvers into a latent-factor based QoS predictor which can achieve higher prediction accuracy for industrial applications [15]. The above studies come from actual QoS data and emphasize the role of different measure factors in software performance analysis and prediction. When conducting a structural reliability analysis of a software project, an important revelation is whether the actual software metrics data can be effectively used. In order to avoid state explosion, the trade-off of module granularity should be based on actual project scale and difficulty of obtaining metrics.

Recent reliability empirical studies most use traditional software-reliability-growth-models (SRGMs) which focus on failure data from the test stage. Luan and Huang [16] present a Pareto distribution of faults in large-scale open source projects for better prediction fitting curve. Sukhwani *et al*. [17] apply SRGMs to NASA's SpaceFlight software to analysis of relevant experience information in the software development process and version management. Aversano and Tortorella [18] propose a reliability evaluation framework for free/open source projects. It was applied to evaluate quality of an open source ERP (enterprise resource planning) software based on a project bug report. Honda *et al*. [19] discuss the effect of two types of measurement units — hourly time and calendar time — in reliability prediction of industrial software systems. Tamura and Yamada [20] present a hierarchical Bayesian model based on fault detection rate around a series of open source solutions and address the impact of conflict behavior of components when dealing with system reliability. This work considers structural information at the component-level, but only takes into account the noise caused by these conflicts in the early stage of the fault detection curve due to the missing discussion of the structural analysis.

The above works cannot be employed in the early stage of development since they require failure data obtained from software tests. Failure data can be attributed to the scope of software metrics. Besides, another type of software metrics has been applied to reliability analysis and defect prediction of actual projects. Complexity measurement data, which has low collection cost, can also be utilized in cognitive modeling [21]. Shibata *et al*. [22] combine a cumulative discrete-rate risk model with time-related measurement data, and prove that the new model is equivalent to a generalized fault detection process whose goodness of fit and predictive performance are better than the popular NHPP SRGMs. Kushwaha and Misra [23] consider the importance of the cognitive measure of complexity and use it in a more reliable software development process. Chu and Xu [24] present a general functional relationship between complexity metrics and software failure rate, which can be used to predict reliability on exponential SRGMs. Bharathi and Selvarani [25] calculate the reliability influence factors separately for several object-oriented design metrics, and finally, deduce the reliability formula of the class-level granularity by merging these factors through an aggregation strategy. This work is a beneficial trial for evaluating software reliability during the design phase. D'Ambros *et al*. [26] compare the performance of several software defect prediction methods and explain the factors of threat validity in practical applications. This work is generally based on static source code metrics and dynamic evolution metrics. Zhang *et al*. [27] group existing defect prediction models based on software metrics into four categories. Through verification of a large number of open source projects, they describe how to aggregate these metrics in order to achieve a significant effect on predictive performance and recommend the simple and efficient aggregation scheme of summation.

However, measurement and SRGM-based empirical studies have several limitations: 1) they cannot calculate the reliability in the early design stage of software system; 2) they are not structure-based methods, and cannot assist in architecture design and optimization. Markovian models such as DTMC can work for this, but they usually have great difficulty in practical application due to lack of real data. We have seen the combination of software metrics with traditional reliability models and detect-prediction methods [22], [24], [25]. We aim to incorporate software metrics into early modeling for software reliability in order to effectively quantify system performance. Our empirical study focuses on the following two research questions:

RQ1: From the perspective of early reliability analysis, what kind of structure granularity is appropriate and how does one use software metrics?

RQ2: How does one evaluate overall system reliability in practical engineering?

The remainder of this paper is organized as follows. Section II introduces the reliability model DTMC and the formal method we used for model construction and calculation. Section III describes the experimental methods of this paper, including object selection, metric data processing and aggregation. The results of this study are presented in Section IV. We evaluate the performance of our approach against other models in Section V, and discuss the threats to validity of our work in Section VI. Conclusions are drawn and future work is described in Section VII.

II. Related Work

In this section, we introduce the typical early reliability model first, and then present the method we utilized to apply the model.

A. Structure-Based Reliability Model

In the early stage of software development, the traditional reliability model is not available due to lack of failure data. A class of structure-based models work for this [1]–[3]. The most popular one is the DTMC (discrete time Markov chain) model [6]. Here is an example of the ESA's (European space agency) control system software [28]. It contains four main components (N_1 to N_4), and the control transfer between components as shown in Fig. 1.

Correspondingly, we have a one-step stochastic transfer matrix as follows:

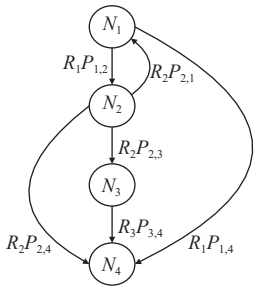


Fig. 1. The ESA software architecture.

$$\boldsymbol{Q} = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \\ N_4 \end{array} \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_4 \end{array} \\ \begin{bmatrix} 0 & R_1P_{1,2} & 0 & R_1P_{1,4} \\ R_2P_{2,1} & 0 & R_2P_{2,3} & R_2P_{2,4} \\ 0 & 0 & 0 & R_3P_{3,4} \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}. \quad (1)$$

The entry $Q_{i,j}$ represents the transition probability from state i to j in the Markov process. Here, $Q_{i,j} = R_1P_{1,2}$, reflects the probability that the system execution state is transferred from N_1 to N_2 in one step, which equals to the product of reliability R_1 and the transfer probability $P_{1,2}$. R_1 expresses the probability of executing N_1 successfully. Each component N_i may fail with the probability $1–R_i$, and each component failure causes system failure. This is a failure independent hypothesis for this model.

The power $\boldsymbol{Q}^n$ of the matrix $\boldsymbol{Q}$ is defined as an n-step stochastic transfer matrix whose entry $Q_{i,j}{}^n$ reflects the transition probability from state i to j in n steps. The Neumann series of matrix $\boldsymbol{Q}$ is

$$\boldsymbol{S} = \boldsymbol{I} + \boldsymbol{Q} + \boldsymbol{Q}^2 + \boldsymbol{Q}^3 + \cdots = \sum_{k=0}^{\infty} \boldsymbol{Q}^k \quad (2)$$

where $\boldsymbol{I}$ is the identity matrix. Let N_1 be the starting module and N_4 the ending module. The entry $S_{1,4}$ in matrix $\boldsymbol{S}$ describes the transition probability from N_1 to N_4 through all possible steps. Then the system reliability can be expressed as the probability of successfully reaching N_4 and successfully executing N_4, which can be calculated by

$$R_{sys} = S_{1,4}R_4. \quad (3)$$

This model emphasizes the influence of structural changes on overall reliability from the perspective of control flow. Subsequent improvements have been proposed on this basis.

B. Algebraic Method

The DTMC model is built around the control transfer relationship between components. However, the reality is that designers seldom use it to represent the system architecture, and module developers only generate control flow graphs at the method-level. For this we have provided an easy-to-use method in our previous studies [29]. In general, we use an algebraic paradigm instead of graphical expression for higher abstraction. The transfer relationship $N_1 \rightarrow N_2$ in Fig. 1 is expressed as $N_1 \oplus N_2$, where the operator $\oplus$ denotes that the basic control transfers between components are motivated [30]. So Fig. 1 can be expressed as a set as follows:

$$\{N_1 \oplus N_2, N_1 \oplus N_4, N_2 \oplus N_1, N_2 \oplus N_3, N_2 \oplus N_4, N_3 \oplus N_4\}. \quad (4)$$

This solves the problem that where the system structure is too complicated for graphic expressions. Especially, when components are designed as a coupling substructure, such as parallel, it is naturally solved by adding an operator in algebra. We can even express the nesting of structures by bracketing them.

We propose a parser which left-to-right scan and rightmost derivate (LR) each expression in the collection as above. The following functions are used in the parser algorithm:

$$\hat{f} : (C \times \cdots \times C) \rightarrow (\mathcal{S} \times \cdots \times \mathcal{S}) \quad (5)$$

where C is a collection of components, $\mathcal{S}$ is a collection of state nodes. This series of $\hat{f}$ for different algebraic expressions are used to generate system state nodes which contains reliability attributes, and mark them on the transition matrix $\boldsymbol{Q}$. The elements in the matrix are constantly updated in the scan. When the scan is complete, the matrix can be directly applied to (2).

This formal method can be easily instrumentalized to facilitate engineers. In the simulation study, it performed well. We summarize the above method as a process framework, as shown in Fig. 2.

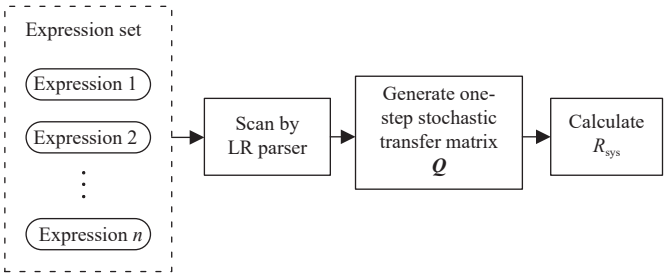


Fig. 2. A framework for automatically applying DTMC.

III. EXPERIMENTAL DESIGN

Properties of the research object and its experimental data are explained in detail. Some methods are applied to reasonably preprocess and aggregate metric data. And the improved framework is used to calculate the overall reliability.

A. Research Object

We select the open source project jEdit [31] as the research object. jEdit is a mature programmer's text editor with hundreds of person-years of development behind it. It is written in Java and runs on any operating system with Java support. This project is in proper scale, and it is representative for development technology.

The current version is set as jEdit 4.3. It is because that the metric data of jEdit in the PROMISE repository [32] contains version 3.2.1 to version 4.3. Although version 4.3 is not the last stable version, it does not affect the universal property of this study since there is no fundamental change in higher versions. We consider ourselves as developers and designers, so we can get the necessary information of structure from the

TABLE I
THE STRUCTURAL INFORMATION OF JEDIT 4.3

Module	Mark	Description	Files	Classes
browser	N_1	File system browser	10	10
bsh	N_2	Bean shell	115	106
buffer	N_3	Buffer event listeners	18	18
bufferio	N_4	I/O request for buffering	6	6
bufferset	N_5	Set of buffer	4	4
gui	N_6	Various GUI controls and dialog boxes	86	88
help	N_7	Help viewer	8	8
indent	N_8	Indent rules and actions	9	9
input	N_9	Input handler	4	4
io	N_{10}	Virtual file system and multi-threaded I/O	18	18
menu	N_{11}	Menu function	13	13
msg	N_{12}	EditBus messages	17	17
options	N_{13}	Global options dialog box panes	25	28
pluginmgr	N_{14}	Plugin manager	10	10
print	N_{15}	Printing	3	3
proto	N_{16}	URL protocol handler	2	2
search	N_{17}	Search and replace classes	19	19
syntax	N_{18}	Syntax highlighting engine	14	14
textarea	N_{19}	An API partition for the standalone text area	38	38
visitors	N_{20}	Visitor pattern	3	3
util	N_{21}	Utility classes that do not depend on jEdit itself	18	18
xml	N_{22}	An obsolete and deprecated XML parser	4	4
core	N_{23}	jEdit's core classes	52	52
Total	**23**	–	**496**	**492**

design document or source code. This information includes packages, files, and classes, as listed in Table I.

Here we define the granularity of structural module analysis at the package-level, and the corresponding level can be found in other language environments. It needs to be based on the previous version when we are seeking detailed module information at the design stage. It usually works because of the limited changes in modules between versions. As the coding continues, we can continuously adjust the changes to the files and classes of the modules. The final structure information of version 4.3 is as described in Table I, and the metrics are based on this.

B. Metric Data Processing

The PROMISE library includes nearly thirty metrics for jEdit. We use three main categories — traditional metrics, object-oriented (OO) metrics, and process metrics — to describe the metric data. The data are summarized at the method-level, class-level and file-level respectively [33], and the developer can easily collect them via the specified tools. At the method-level, the number of lines of code (LOC) and the cyclomatic complexity (CC) [34] are still suitable for code analysis inside a class, which existed before object-oriented programming appeared. The CK set [35] has a wide range of applications, but there are also metrics that emphasize perspectives such as encapsulation, coupling, etc. [36]. The eight metrics recommended by Moser *et al*. [37] have typical process characteristics, and are further improved in the MJ set [38].

Simple data sampling is not used in this paper. For early reliability analysis, the initial metric data needs to be cleaned to highlight the structural characteristics of the metric elements. We use the following method:

1) The process metric elements (from Moser and MJ set) do not apply to static analysis for the target version. This work only uses traditional metrics and OO metrics.

2) The remaining 20 metrics are divided into five categories: complexity, coupling, cohesion, inheritance and size. We use Spearman's coefficient [39] to measure correlation in 5 subsets. The data is not required to follow any particular distribution in the state of Spearman correlation, and it ranges from –1 to +1 where a larger absolute value indicates the stronger correlation. Table II shows the calculation results (e.g., the coupling metrics).

Absolute values of more than 0.5 have relevance and values of more than 0.8 have high correlation. Observe that there are redundant metrics that can be removed to further simplify calculations. According to the CFS algorithm of attribute selection [40], one metric with relatively high relevance is

TABLE II
CORRELATION OF THE COUPLING METRICS

	CBO	CA	CE	IC	CBM
CBO	–	0.703	0.684	0.157	0.171
CA	0.703	–	0.142	–0.061	–0.049
CE	0.684	0.142	–	0.351	0.363
IC	0.157	–0.061	0.351	–	0.985
CBM	0.171	–0.049	0.363	0.985	–

selected as a main feature in each category. The maximum cyclomatic-complexity (MAX_CC), coupling between object classes (CBO), cohesion among methods (CAM), depth of inheritance tree (DIT), and response for a class (RFC) are finally selected elements as representative for the five categories separately. It ensures that subsequent calculations which can characterize 5 different characteristics instead of examining correlations across all metrics. Table III lists the data we actually used in this research when deal with the browser module in Table I.

TABLE III
THE METRIC DATA USED IN THIS RESEARCH

Class name (Prefix:org.gjt.sp.jedit. browser.)	MAX _CC	CBO	CAM	DIT	RFC
BrowserCommandsMenu	5	16	0.314	5	56
BrowserIORequest	8	10	0.265	2	29
BrowserListener	1	4	1	1	2
BrowserView	9	28	0.148	5	106
FileCellRenderer	9	15	0.243	6	52
VFSBrowser	21	75	0.114	5	291
VFSDirectoryEntryTable	40	27	0.158	5	144
VFSDirectoryEntryTable-Model	6	13	0.379	2	47
VFSFileChooserDialog	13	21	0.289	7	110
VFSFileNameField	21	11	0.44	7	57
Avg	13.3	22.0	0.335	4.5	89.4

C. Aggregation Scheme

The metric data are only collected at the class-level. It is necessary to propose a descriptive scheme to aggregate the final value of software system reliability. We suggest a two-step frame and the details are presented as follows.

Step 1: To establish a functional relationship between metrics and degree of reliability in order to calculate module reliability value.

We define the degree of reliability influence, which is similar to the definition in [25]. Let RI_{mi} represent influence of the metric scalar m_i. It is calculated by

$$RI_{m_i} = \left[1 - \left(\frac{dpc}{mc}\right)\right] \times 100\% \tag{6}$$

where *dpc* indicates the number of defect prone classes at m_i, and *mc* indicates the number of classes contained in one module. Notice that *RI* is only calculated in one module, since the coding style of each module may not be consistent. The range of scalar m_i includes all samples of the specified metric element. Table IV lists m_i, *dpc*, and RI_{mi} of 5 metric elements in the module browser, when $mc = 10$.

It is observed from Table IV that *dpc* is counted for each m_i per elements, and each m_i value is a sample of the metric space. The counting method of *dpc* is summarized as: If the class, which is associated with the specific value of m_i at one metric element, a) is submitted with a bug in the previous version, $dpc \leftarrow dpc + 1$; b) or is newly developed and m_i exceeds threshold, $dpc \leftarrow dpc + 1$.

The thresholds of RFC(222), CBO(24), and DIT(6) refer to the NASA standard [41]. The threshold of CC(15) is derived from [34], and the threshold of CAM(0.50) comes from [38].

In general, lower metric value means higher reliability. But it cannot be concluded that there is a linear correlation between metric value and reliability influence. We use polynomial regression to estimate the relationship between the two, because of changes of software reliability mostly in the form of curves. As an example, the functional estimation equations of the browser module are

$$\begin{aligned} RI_{\mathrm{RFC}} &= 102.295 - 0.127a \\ RI_{\mathrm{MAX\_CC}} &= 100.652 - 0.927b + 0.017b^2 \\ RI_{\mathrm{CAM}} &= 86.622 + 37.292c - 24.085c^2 \\ RI_{\mathrm{CBO}} &= 104.712 - 0.698d + 0.007d^2 \\ RI_{\mathrm{DIT}} &= 116.182 - 18.103e + 2.167e^2 \end{aligned} \tag{7}$$

where a, b, c, d, e are the independent variables in each fitting function. According to the average value of metrics listed in Table IV, the five degree of reliability influence in the browser module can be calculated as: $RI_{\mathrm{RFC}}(89.4) = 90.941$, $RI_{\mathrm{MAX\_CC}}(13.3) = 91.330$, $RI_{\mathrm{CAM}}(0.335) = 96.414$, $RI_{\mathrm{CBO}}(22.0) = 92.744$, $RI_{\mathrm{DIT}}(4.5) = 78.600$.

Aggregation strategy can significantly alter correlations between software metrics and the defect count. Zhang *et al.* [27] indicate that the summation strategy can often achieve the best performance when constructing models predict defect rank or count. Although the perspectives of defect prediction and reliability evaluation are different, they are effectively the same in terms of the use of metrics. The difference is that the former focuses on the situation after the current version, and the latter focuses on the current developing version. Therefore, in this paper, we use the summation strategy to calculate the reliability degree of the individual module according to the reliability influence of each factor. It is calculated as

$$R_{\mathrm{module}} = \sum_i \left(r_i \times RI_{\mathrm{metric}_i}(\overline{m}_i)\right) \tag{8}$$

where r_i is weight of the ith metric element. Since the five metrics used in this paper are pre-screened to represent a logical category, they are considered equally important. Let $r_i = 1/n = 1/5$, where reliability of the browser module is 90.006(%) after the weighted summation.

Step 2: To establish the structural reliability model according to R_{module}.

In the software design and development phase, reliability engineering often needs to be carried out by architects and

TABLE IV
THE *RI* STATISTICS IN THE BROWSER MODULE

Metric element	m_i	*dpc*	RI_{mi}	m_i	*dpc*	RI_{mi}
RFC	2	0	100	57	0	100
	29	0	100	106	1	90
	47	1	90	110	1	90
	52	0	100	144	1	90
	56	0	100	291	1	90
MAX_CC	1	0	100	9	1	90
	5	0	100	13	1	90
	6	1	90	21	1	90
	8	0	100	40	1	90
CAM	0.114	1	90	0.289	1	90
	0.148	1	90	0.314	0	100
	0.158	1	90	0.379	1	90
	0.243	0	100	0.44	0	100
	0.265	0	100	1	0	100
CBO	4	0	100	16	0	100
	10	0	100	21	1	90
	11	0	100	27	1	90
	13	1	90	28	1	90
	15	0	100	75	1	90
DIT	1	0	100	6	0	100
	2	1	90	7	1	90
	5	3	70			

coders. The module developer calculates R_{module} by counting the related metrics. One can gather this information of all modules to aggregate into system reliability.

Here we use the formal method presented in Section II-B to establish the DTMC reliability model quickly and accurately. The method is based on algebraic expressions, which can accurately express the control transfer between modules instead of using graphical representation. In addition to the value of R_{module}, module developers are required to submit related expressions based on their understanding of the business. For the browser module (N_1), the developers need to submit:

i) R_{N1}= 90.006%;

ii) $N_1 \oplus N_{21}$.

Note that the expression $N_1 \oplus N_{21}$ replaces the directed arc of graph to describe the control transfer relationship between N_1 and N_{21}. It indicates that further processing of the file system will go to the utility module (N_{21}).

A module developer can separately submit expressions that confirm design intent, which formally starts with the developing module and links all possible next modules in the control flow. In some cases, architects can also submit or modify expressions based on overall understanding. As a description of the structure, algebraic expressions are precise and unambiguous. They are lightweight and easy to use for engineers. One can do this at the same time as designing and coding, without any distractions in describing the whole system.

Eventually, an expression set can be collected which contains two key parameters required for the DTMC model — R_i and P_{ij}. R_i comes from the submitted information which is usually attached to a concrete expression. Transfer probability P_{ij}, which is indicated by $N_i \oplus N_j$, is equally divided by all possible transfers from module N_i. The LR parser presented in Section II-B can calculate P_{ij} automatically during scanning.

Finally, as an improvement to Fig. 2, the new framework is proposed as Fig. 3.

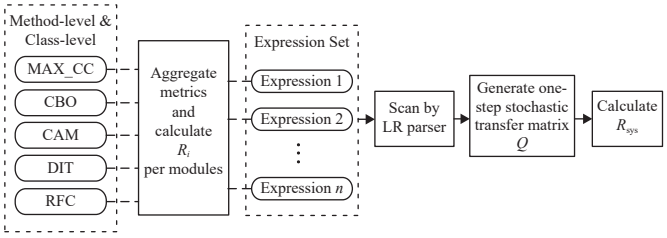


Fig. 3. The improved framework for empirical research.

IV. RESULTS

This section provides our experimental results. We focus on findings from actual application of methods in Section III, and answer the research questions presented in Section I.

RQ1: How to determine the appropriate structure granularity in order to apply the corresponding metrics to reliability analysis?

In general, software design has the characteristics of modularization, which is advantageous to the maintenance of projects and the deployment of resources. Early reliability analysis can only start with the structure of the software system. The structural characteristics are represented by the relationships between modules. This requires selecting the appropriate module granularity. In this paper, we suggest that Java OO projects should be analyzed at the package-level. The corresponding granularity of other types of OO projects can be found in the directory structure and design documents.

Table I lists the module division of jEdit. Some modules of version 4.3 do not exist in previous versions, and some have changed in later versions. However, analysis at the package-level can cover all relevant metrics collected at the method- or class-level. Different packages show significant differences in metrics due to the diversity of functionality and developer. We randomly choose eight out of 23 modules, and present boxplots of them under the five selected metrics. As shown in Fig. 4, the distributions of metric data are obviously different, and most of them are different from the entire distribution. It shows that the coding style of each module is different, i.e., the quality of each module in the same software is also different. Furthermore, the life cycle and application scope of many components (packages in java projects) goes beyond the project itself. E.g., "util" and "xml", which are independent of jEdit's main business, came from or could be used for other projects. Therefore, we think that for each different module, the reliability influence from its metric data should be analyzed separately. And then the reliability data of all

modules should be integrated with a specific structural analysis method.

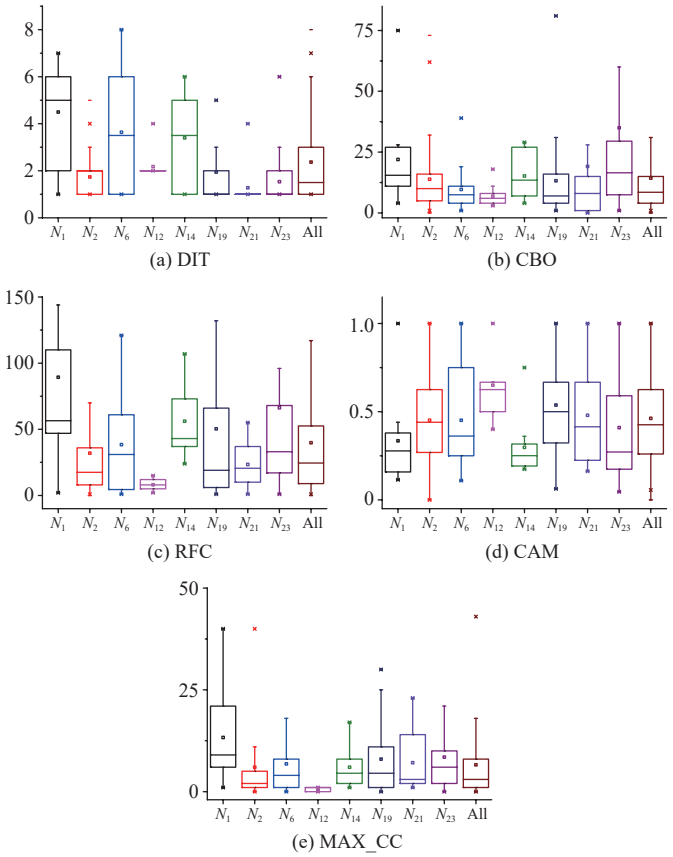


Fig. 4. Boxplots of 5 metric values resourced from eight modules and whole.

RQ2: How to evaluate the reliability of a software?

We calculate the reliability value of a single module based on (6) and (8). Table V lists the reliability degree for all modules in 4 versions of the jEdit project. Note that version 4.3 is the last version in PROMISE repository, and it is also the one set for the current review. The first version 3.2.1 is discarded because we need to trace back a version for counting *dpc* in (6).

Fig. 5 presents the boxplots of the numerical results in Table V. It shows that statistically, the overall distribution gradually increases as the version changes. It can be seen from the diagram that most module reliability values increase as the version changes. This is in line with the expectation that the module quality will improve as more bugs are fixed. From the perspective of structural analysis, we still want to estimate a specific value to reflect the overall reliability of software.

With version 4.3 as the goal, module developers should present the relationship between this module and other modules in addition to R_{module}, which can be described conveniently by algebraic expressions. Then, the expression set of $\{N_{12}\oplus N_3,\ N_{12}\oplus N_6,\ N_{12}\oplus N_{19},\ \ldots\}$ can be collected. Expression $N_{23}\oplus \underline{C}$ should be added into the collection in order to facilitate model calculation, which means the requirement of business termination is only initiated by the core module N_{23}. $\underline{C}$ is the ending node and its reliability value is 100%. The starting module is N_{12}, which means that the message occurs when the business is coming.

TABLE V
THE R_{MODULE} VALUES IN DIFFERENT VERSIONS OF JEDIT

Module	The value of R_{module} (%)			
	v4.0	v4.1	v4.2	v4.3
browser	86.755	89.104	89.303	90.006
bsh	87.667	88.482	94.358	96.385
buffer	87.567	86.415	92.883	95.812
bufferio	–	–	–	95.521
bufferset	–	–	–	89.871
gui	86.238	85.116	89.552	91.350
help	–	86.633	89.569	90.018
indent	–	–	–	97.656
input	–	–	–	98.142
io	89.008	89.664	90.322	97.765
menu	–	–	94.574	97.161
msg	88.089	89.528	93.147	96.502
options	89.885	91.207	92.422	96.352
pluginmgr	90.125	91.897	95.322	97.151
print	88.252	89.128	91.082	94.056
proto	86.997	85.328	90.998	94.106
search	89.231	90.562	90.484	96.531
syntax	84.003	85.662	88.557	94.481
textarea	84.712	83.563	90.245	92.266
visitors	–	–	–	97.175
util	91.367	91.251	93.699	96.623
xml	90.574	92.236	90.011	95.754

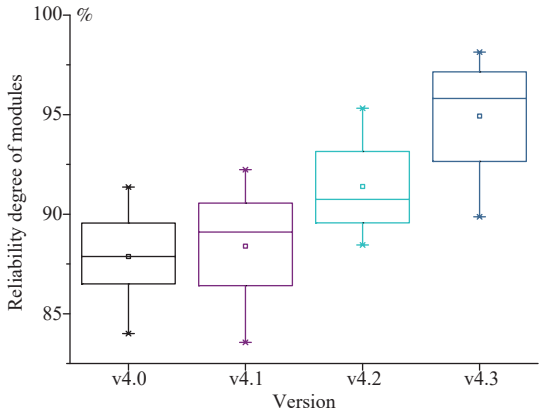


Fig. 5. Boxplot of reliability degree in all modules among four versions.

We also set up an expression set for three other versions, and use the Fig. 3 framework to automate the overall software reliability calculation. Fig. 6 shows the results. To illustrate the effectiveness of the proposed method, two traditional growth models: the classical G-O model [8] and Huang's test-effort model [42] are used as comparisons. In this work, we follow the common practice in empirical research of SRGMs [16]–[19], where the failure data is extracted from the bug

reports. This is under basis of the following assumptions: 1) Confirmed bugs cause failures; 2) Each failure is independent; 3) Fixing bugs will not introduce new bugs. According to the reports obtained from the official website of jEdit [31], version 4.0 has 226 closed bugs within 215 days since the previous release. The G-O model use a statistical process based on this failure intensity, and the prediction result at the release time of jEdit 4.0 is 88.736%. Version 4.1 has 167 closed bugs within 407 days, and the prediction result is 89.413%. The data for versions 4.2 and 4.3 show that there are 106 bugs within 557 days and 12 bugs within 1848 days. The prediction results are 92.185% and 96.587%,respectively. Furthermore, as suggested in [42], the cumulative interval is used here as the test-effort which is available from the detailed entry of each bug in the report. The prediction results of Huang model across the four versions are 90.156%, 91.142%, 92.868% and 96.224%, respectively.

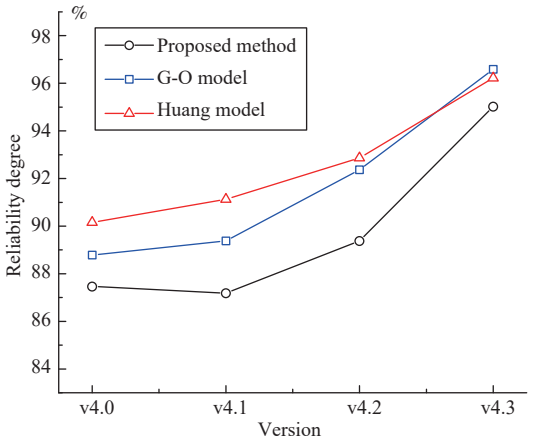


Fig. 6. Reliability trends on four versions of jEdit.

V. Discussion

A. Characteristics of Early Reliability Modeling

Unlike SRGMs, early reliability modeling only relies on structural analysis of software system. As mentioned in the first section, the modeling parameters come from a single software module and the relationship between modules. Therefore, it is meaningless to compare the predictive performance with a class of models based on failure data. Most reliability-prediction models, such as SRGMs, are utilized to optimize the release strategy to achieve a balance between quality and test costs. But the significance of the early model is to optimize the structure and guide the test.

We show the trend of changes in the results of the proposed method in Fig. 6, and also use two representative models to illustrate the correctness. It should be explained that the calculation of each version of this method can be performed at the design stage. And based on the framework in Section III-*C*, the calculation process is automatically complemented by tools. This means that the reliability evaluation can be carried out at any time with structural design changes. It helps decrease the difficulty of applying the structural reliability model into practice.

As shown in Fig. 6, the numerical values evaluated by the proposed method (87.267%, 87.184%, 89.381%, 94.223%) are lower than by the traditional models. This is because calculations based on metric data can magnify the impact of defect-prone classes. For the calculation of defect tendency, the method of this paper tends to be conservative. Correspondingly, the SRGMs which are based only on test data assume that the reliability curve increases with bugs fixing. Their evaluations are relatively optimistic in general.

B. Sensitivity Analysis

The influence of software local structure on reliability can be analyzed through sensitivity. This effect can only be manifested in the computation of the structural reliability model. As shown here jEdit 4.0 is upgraded to 4.1. The result, which differs from the traditional models', appears to be slightly reduced. By partially fixing pre-existing defects, new versions can also have reduced reliability with the introduction of new modules and features (such as N_7 in jEdit 4.1).

We use the criticality to analyze sensitivity. It is computed by the following formula:

$$C_i = \frac{\Delta R_{\text{sys}}}{\Delta R_{\text{i}}} \tag{9}$$

where ΔR_i is the reliability increment of module N_i, ΔR_{sys} is the overall reliability increase corresponding to this increment. In fact, when the ΔR_i is very small, we use C_i instead of the partial derivative, i.e., the criticality of module N_i is an approximate value of the module sensitivity. However, in terms of computational complexity, computing criticality is much better than computing sensitivity. The sensitivity (partial derivative) on R_i can only be solved based on cofactor expansion, which leads to a geometric increase in computational time and space when the total number of modules grows. In contrast, (9) can be calculated automatically by the proposed framework in Fig. 3.

For jEdit 4.3, we set a small negative increment of 0.005 for all module reliability. Fig. 7 shows the C_i corresponding to each change of ΔR_i. It can be seen that the reliability sensitivity of core module N_{23} and message module N_{12} are significantly higher than others. This conforms to the expectation of actual control flow at design stage. And this means that later testing on such modules will help significantly improve the overall reliability of software. However, some modules (such as N_{18}) are not considered to be the important nodes in structure because they only interact with very few modules. These may not be the goals that reliability engineering needs to focus on.

VI. Threats to Validity

1) Project Selection: In this work, the open source project jEdit is selected. In addition to using metrics, we need to analyze the software structure and apply a complex evaluation framework. We only discussed one project because of space constraints.

The threat to our treatment mainly arises from applicability of our method on other OO projects. Metric data can be

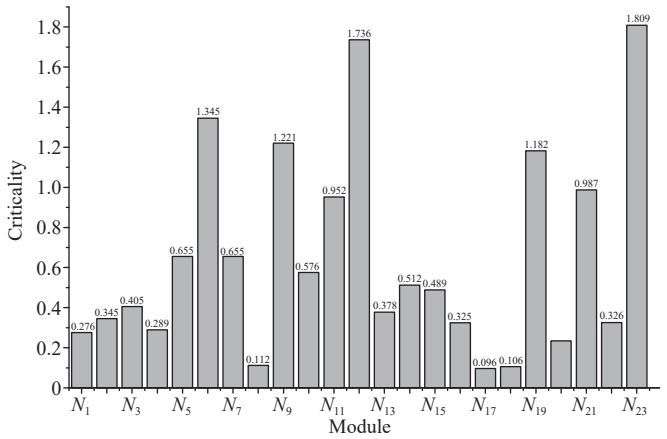


Fig. 7. Analysis of reliability sensitivity of modules in jEdit 4.3.

obtained from public repositories, but also can be directly calculated from the source code. Structural information can be analyzed from design documents and source codes. We believe that the applicability of the proposed method is not a problem. Furthermore, it is recommended that non-java projects should be tested using our approach and the results may be slightly different.

2) Metrics and Techniques Selection: These metrics and its classification we used are popular and commonly investigated in defect prediction literature. With the same kind of metrics, correlation analysis technique is used to eliminate redundant impacts. A study on more correlation analysis techniques is required. In addition, the DTMC model used in this paper is the most important structural analysis model at present. The proposed framework uses formal techniques to show the application of this model. Formal techniques have been fully elaborated and verified in our previous studies. We will provide an open-source implementation of analytical algorithms online. Replication studies using different early reliability models, such as CTMC, to utilize this framework and may prove fruitful.

VII. Conclusion

Although there are many practical difficulties in reliability analysis and evaluation of software projects in early development, the calculation results from the reliability model can provide reference for the optimization of software architecture design. By analyzing the software structure, we find the sensitive modules that affect the overall reliability, which provides a practical basis for subsequent reliability distribution and stress testing. Theoretical research of structure-based software reliability models has been studied extensively, but the biggest obstacle to its further development is determining how to apply it to practical projects.

Our main work is to provide a complete solution for early reliability evaluation. We first give a method for processing metric data and aggregating it into module reliability. Then we propose a process framework, which automatically calculates the overall reliability value based on module parameters and expression forms. Using a series of different methods, our research shows the implementation of a structural reliability model in practical projects. Evaluation of an OO project shows that we can get an approximating result vs. traditional models which are based on software failure data. It provides new ideas and methods for the empirical research of reliability.

The work is original and operable in the selection and aggregation of software metric data and the practice of structural analysis model. The next steps of research will be continued in two areas:

1) Consider establishing a functional relationship between metrics and reliability on more metric elements. The fitting method of the function is not limited to one but also, selects the appropriate method according to different modules and developers, such as Stepwise Regression and Ridge Regression.

2) Defect prediction Technology continues to show new results. In addition to the traditional methods, a method using random neural networks (RNN) has recently been proposed, and the results are good. This prediction model is established on the time axis and describes the evolution of each software module. We will consider the use of better-performing solutions in the prediction of defect-prone classes to improve the reliability evaluation of modules.

References

[1] F. Febrero, C. Calero, and M. Á. Moraga, "A Systematic Mapping Study of Software Reliability Modeling," *Inform. Software Tech.*, vol. 56, no. 8, pp. 839–849, 2014.

[2] H. Mei, G. Huang, L. Zhang, and W. Zhang, "ABC: A method of software architecture modeling in the whole lifecycle," *Science China-Information Sciences*, vol. 44, no. 5, Article No. 564, 2014.

[3] A. D. Plessis, K. Frank, M. Saglimbene, and N. Ozarin, "The thirty greatest reliability challenges," in *Proc. Reliability and Maintainability Symposium*, vol. 94, pp. 1–6, Jan. 2014.

[4] T. Dan, M. Galster, P. Avgeriou, and W. Schuitema, "Past and future of software architectural decisions – A systematic mapping study," *Inform. Software Tech.*, vol. 56, no. 8, pp. 850–872, 2014.

[5] B. Littlewood, "Software reliability model for modular program structure," *IEEE Trans. Reliability*, vol. R–28, no. 3, pp. 241–246, 1979.

[6] R. C. Cheung, "A user-oriented software reliability model," *IEEE Trans. Software Engineering*, vol. SE–6, no. 2, pp. 118–125, 1980.

[7] J. C. Laprie, "Dependability evaluation of software systems in operation," *IEEE Trans. Software Engineering*, vol. SE–10, no. 6, pp. 701–714, 1984.

[8] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Reliability*, vol. R–28, no. 3, pp. 206–211, 1979.

[9] W. B. Zheng, M. C. Zhou, L. Wu, Y. N. Xia, X. Luo, S. C. Pang, Q. S. Zhu, and Y. Q. Wu, "Percentile performance estimation of unreliable IAAS clouds and their cost-optimal capacity decision," *IEEE Access*, no. 5, pp. 2808–2818, 2017.

[10] J. Li, X. Luo, Y. N. Xia, Y. K. Han, and Q. S. Zhu, "A time series and reduction-based model for modeling and QoS prediction of service compositions," *Concurrency and Computation*: *Practice and Experience*, vol. 27, no. 1, pp. 146–163, 2015.

[11] Y. N. Xia, X. Luo, J. Li, and Q. S. Zhu, "A Petri-net-based approach to reliability determination of ontology-based service compositions," *IEEE Trans. Systems*, *Man*, *and Cybernetics*: *Systems*, vol. 43, no. 5, pp. 1240–1247, 2013.

[12] S. S. Gokhale, "Architecture-based software reliability analysis: overview and limitations," *IEEE Trans. Dependable and Secure Computing*, vol. 4, no. 1, pp. 32–40, 2007.

[13] Y. N. Xia, M. C. Zhou, X. Luo, Q, S. Zhu, J. Li, and Y. Huang, "Stochastic modeling and quality evaluation of Infrastructure-as-a-Service clouds," *IEEE Trans. Autom. Science and Engineering*, vol. 12, no. 1, pp. 162–170, 2015.

[14] X. Luo, M. C. Zhou, Z. D. Wang, Y. N. Xia, and Q. S. Zhu, "An effective scheme for QoS estimation via alternating direction method-based matrix factorization," *IEEE Trans. Services Computing*, vol.12, no.4, pp.503–518, 2019.

[15] X. Luo, M. C. Zhou, S. Li, Y. N. Xia, Z. H. You, Q. S. Zhu, and H. Leung, "Incorporation of efficient second-order solvers into latent factor models for accurate prediction of missing QoS data," *IEEE Trans. Cybernetics*, vol. 48, no. 4, pp. 1216–1228, 2018.

[16] S. P. Luan, and C. Y. Huang, "An improved Pareto distribution for modelling the fault data of open source software," *Software Testing, Verification and Reliability*, vol. 24, no. 6, pp. 416–437, 2014.

[17] H. Sukhwani, J. Alonso, K. S. Trivedi, and I. Mcginnis, "Software reliability analysis of nasa space flight software: A practical experience," in *Proc. IEEE Int. Conf. Software Quality, Reliability and Security*, vol. 3, pp. 386–397, 2016.

[18] L. Aversano, and M. Tortorella, "Analysing the reliability of Open Source software projects," in *Proc. 10th Int. Joint Conf. Software Technologies*, pp. 348–357, 2016.

[19] K. Honda, N. Nakamura, H. Washizaki, and Y. Fukazawa, "Case study: Project management using cross project software reliability growth model," in *Proc. IEEE Int. Conf. Software Quality*, *Reliability and Security Companion*, pp. 41–44, 2016.

[20] Y. Tamura and S. Yamada, "Reliability analysis considering the component collision behavior for a large-scale open source solution," *Qual. Reliab. Eng. Int.*, vol. 30, no. 5, pp. 669–680, 2014.

[21] L. Fiondella, A. Nikora, and T. Wandji, "Software reliability and security: challenges and crosscutting themes," in *Proc. IEEE Int. Symposium Software Reliability Engineering Workshops*, pp. 55–56, 2016.

[22] K. Shibata, K. Rinsaka, and T. Dohi, "Metrics-based software reliability models using non-homogeneous poisson processes," in *Proc. Int. Symposium Software Reliability Engineering*, IEEE Computer Society, pp. 52–61, 2006.

[23] D. S. Kushwaha and A. K. Misra, "Cognitive complexity metrics and its impact on software reliability based on cognitive software development model," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 2, pp. 1–6, 2006.

[24] Y. M. Chu and S. Y. Xu, "Exploration of complexity in software reliability," *Tsinghua Science and Technology*, vol. 12, no. S1, pp. 266–269, 2007.

[25] R. Bharathi and R. Selvarani, "A framework for the estimation of OO software reliability using design complexity metrics," in *Proc. Int. Conf. Trends in Autom. Communications and Computing Technology*, pp. 1–7, 2016.

[26] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4–5, pp. 531–577, 2012.

[27] F. Zhang, A. E. Hassan, S. Mcintosh, and Y. Zou, "The use of summation to aggregate software metrics hinders the performance of defect prediction models," *IEEE Trans. Software Engineering*, vol. 43, no. 5, pp. 476–491, 2017.

[28] K. Goseva-Popstojanova, A. P. Mathur, and K. S. Trivedi, "Comparison of architecture-based software reliability models," in *Proc. Int. Symposium Software Reliability Engineering*, *IEEE Computer Society*, vol. 7, pp. 22–31, 2001.

[29] J. Zhang, Y. Lu, and G. L. Liu, "Algebraic approach of software reliability estimation based on architecture analysis," *Systems Engineering and Electronics*, vol. 37, no. 11, pp. 2654–2662, 2015.

[30] H. Q. Zhao and J. Sun, "An algebraic model of Internetware software architecture," *Science China-Information Sciences*, vol. 43, no. 1, pp. 161–177, 2013.

[31] S. Pestov, "JEdit programmer's text editor (2018)," [Online]. Available: http://www.jedit.org/main.html, Accessed on: Mar. 16, 2018.

[32] G. Boetticher, T. Menzies, and T. Ostrand, "Tera-PROMISE: Welcome to one of the largest repositories of SE research data (2018)," [Online]. Available: http://openscience.us/repo/index.html, Accessed on: Mar. 22, 2018.

[33] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[34] T. J. Mccabe, "A complexity measure," *IEEE Trans. Software Engineering*, vol. SE–2, no. 4, pp. 308–320, 1976.

[35] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Engineering*, vol. 20, no. 11, pp. 197–211, 1994.

[36] D. P. Darcy and C. F. Kemerer, "OO metrics in practice," *IEEE Software*, vol. 22, no. 6, pp. 17–19, 2005.

[37] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. ACM/IEEE Int. Conf. Software Engineering*, pp. 181–190, 2008.

[38] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Software Quality Journal*, vol. 23, no. 3, pp. 393–422, 2015.

[39] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 5th ed, Boca Raton, USA: Chapman & Hall/CRC, 2012.

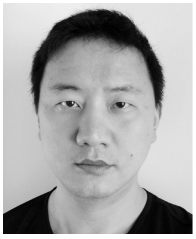
[40] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Attribute selection in software engineering datasets for detecting fault modules," in *Proc. 33rd EUROMICRO Conf. Software Engineering and Advanced Applications*, pp. 418–423, 2007.

[41] G. Brat and A. Venet, "Precise and scalable static program analysis of NASA flight software," in *Proc. IEEE Aerospace Conf.*, pp. 1–10, 2005.

[42] C. Y. Huang, S. Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models," *IEEE Trans. on Reliability*, vol. 56, no. 2, pp. 198–211, 2007.

Jie Zhang received the B.S. degree from Anhui Normal University, in 2002, the M.S. degree from Tongji University, in 2009, and the Ph.D. degree in computer application technology from Hefei University of Technology, in 2019. He is currently an Assistant Professor with the School of Computer and Information, Anhui Normal University. His research interests include system reliability, system safety, software reliability, and trust computing.

Yang Lu received the B.S. degree in electronic engineering from Shanghai Jiao Tong University, in 1988, and Ph.D. degree in computer application technology from Hefei University of Technology, in 2002. He is currently a Professor with the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include computer control, system safety, sensor network, and distributed control system. He has published over 50 research papers in different international journals and conferences.



Ke Shi received the M.S. degree from Anhui University, in 2010. He is currently a Ph.D. candidate at the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include natural language processing, information retrieval, and machine learning.

Chong Xu received the M.S. degree from the School of Computer Science and Information Engineering, Hefei University of Technology, in 2013, where he is currently a Ph.D. candidate. His research interests include software architecture and process algebraic.