Applying Software Metrics to RNN for Early Reliability Evaluation

Jie Zhang
School of Computer and Information
Hefei University of Technology
Hefei, China
zjzj2526@163.com

Yang Lu
School of Computer and Information
Hefei University of Technology
Hefei, China
luyang.hf@126.com

Ke Shi
School of Computer and Information
Hefei University of Technology
Hefei, China
shike@mail.hfut.edu.cn

Chong Xu
School of Computer and Information
Hefei University of Technology
Hefei, China
xuchong15@mail.hfut.edu.cn

***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 models. The structural reliability models are made for this, but most of them remain in the simulation studies because of lack of actual data. In this study, we use software metrics which are collected from actual projects to evaluate the reliability. We use the Recurrent Neural Network to process the metric data to identify defeat-prone classes in one project. A specific strategy is used for aggregating module reliability with the results. Furthermore, we propose a framework which can automatically calculate the overall reliability value by the introduced formal tools. The experimental results of two open-source projects show that reliability analysis at design and development stage can be close to the validity of analysis at test stage through the reasonable application of metric data and related methods.**

Keywords—software reliability, software metrics, software defect, RNN

I. MOTIVATION

Software reliability engineering aims to improve software quality and its role covers all stages of development. Recently more research [1]~[4] believed that reliability evaluation in the early stages has important implications for avoiding the possible revision cost in the later development stage, especially for a class of safety-critical software systems. But most of recent reliability empirical studies still use the traditional growth models (SRGMs) which focus on failure data from test stage. For example, Luan and Huang [5] studied the distribution of faults in large-scale open source projects by using the Pareto distribution to obtain better prediction curve fitting accuracy than the classic Weibull distribution. Sukhwani [6] applied SRGMs to NASA's SpaceFlight software to analysis of relevant experience information in software development process and version management. Aversano and Tortorella [7] gave a reliability assessment framework which was applied to evaluate an open source ERP software based on bug reports. Honda [8] practiced in industrial software projects to discuss the performance of popular SRGMs. Tamura and Yamada [9] built a hierarchical Bayesian model which based on fault detection rate around a series of open source solutions (such as Apache HTTP server, Tomcat, etc.) for reliability analysis.

The above empirical research belongs to the later reliability engineering because based on the test period data. In contrast, early reliability models can work in the design phase to assist in engineering decisions. Typical models include Littlewood's SMP [10], Cheung's DTMC [11] and Laprie's CTMC [12], etc.. They are commonly referred to as structure-based methods which emphasize structural analysis based on a specific granularity. But early models have great difficulty in practical applications. Taking the DTMC model as an example, the two parameters required for modeling: component reliability and control transfer probability among components, are given by simulated cases rather than actual projects [13]. In this research we propose to obtain the necessary information from software codes directly to support the application of early reliability models. From the perspective of improving the engineering process, we discuss the methods of reliability modeling in design and coding period.

In fact software metrics has been applied to the reliability analysis and defect prediction of actual software projects. Shibata [14] incorporated the cumulative discrete-rate risk model with time-related measurement data, and verified that the predictive performance of new model are better than popular NHPP SRGMs'. Chu and Xu [15] gave a general functional relationship between complexity metrics and software failure rate which can be used in the exponential SRGMs. D'Ambros [16] compared the performance of several software defect prediction methods and explained the factors of threat validity in practice. These methods are generally based on static source code metrics and dynamic evolution metrics. In [17], the author summarized the existing defect prediction models based on software metrics into four categories, and indicated how to aggregate them to achieve significant effect on predictive performance.

Fiondella [18] considered that complexity measurement data which has the characteristics of low collection cost and various forms could be utilized in cognitive modeling. Kushwaha and Misra [19] pointed out the importance of the cognitive measure of complexity and uses it in a more reliable software development process. We believe that the cognitive information required for early reliability analysis is already included in the code structure, code metrics, and design documents. We will conduct empirical research on this in order to assist decision-making in the development process.

This work is supported by National Natural Science Foundation (61572167, 51504010) and National Key Research and Development Program (2016YFC0801804) of China.

The rest of this paper is organized as follows. Section II

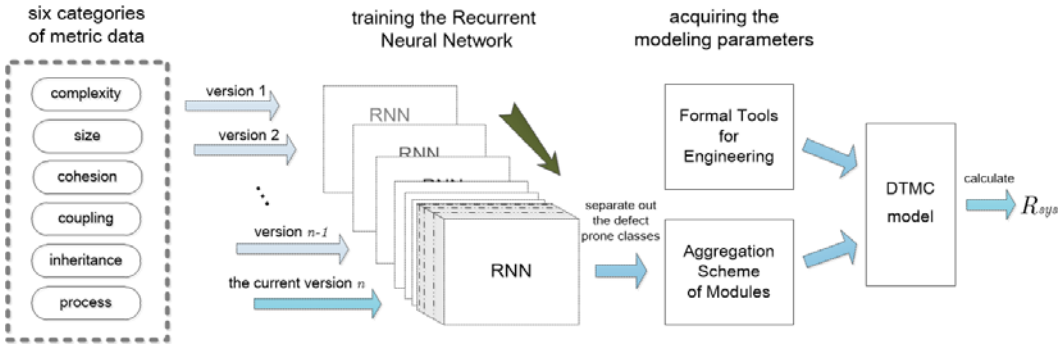


Fig .1. A framework of this paper.

introduces the framework we used for model construction and reliability calculation. Section III gives the experimental methods of this paper, including object selection, metric data processing and aggregation scheme. In Section IV, the results of this research are presented. We evaluate the performance of our approach against other models and discuss the impact of parameters on the results. Conclusions are drawn in Section V.

II. Proposed Approach

A. Framework of This Paper

First, we give our framework as shown in Fig.1 to fully describe the methods used in this article. The actual metric data will be divided into six categories based on their characteristics. And a recurrent neural network (RNN) will be trained on data from historical version in order to separate out the defect prone classes in the current version. The next subsection describes in detail the RNN we used. Then a specified scheme receives class information and aggregates them to module reliability. We also use formal tools to facilitate the application of the DTMC model. We will discuss these in the following sections.

B. Recurrent Neural Network

We use a simple type of RNN which has one hidden layer. Fig.2 describes its main structure.

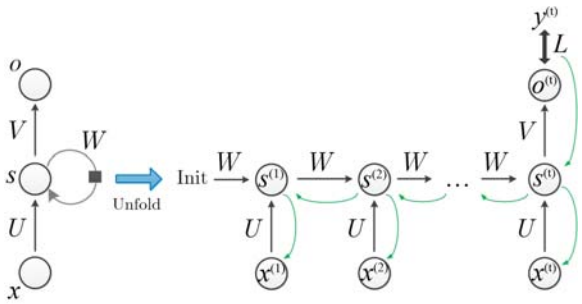


Fig.2. A many to one RNN model. The right is the unfolded form, where U, V and W are the uniform weight matrices. The loss L between the only output $o^{(t)}$ and the goal $y^{(t)}$ is used in the back-propagation for updating model parameters.

As shown, the RNN propagates forward from initial state $\boldsymbol{s}^{(0)}$. The update equations for every time step from 1 to t is

$$\boldsymbol{s}^{(t)}=\tanh\left(\boldsymbol{U}\,\boldsymbol{x}^{(t)}+\boldsymbol{W}\boldsymbol{s}^{(t-1)}+\boldsymbol{b}\right) \quad (1)$$

$$\boldsymbol{o}^{(t)}=\mathrm{sigmoid}\left(\boldsymbol{V}\,\boldsymbol{s}^{(t)}+\boldsymbol{c}\right) \quad (2)$$

where $\boldsymbol{b}$, $\boldsymbol{c}$ are the bias vectors. The Hyperbolic tangent function tanh() is the most commonly used activation function between input and hidden layer. And the logistic function sigmoid() is chose for output function because we only deal with two classification problems here. The loss L is calculated as:

$$L=L^{(t)}=-\boldsymbol{y}^{(t)}\log\boldsymbol{o}^{(t)}-(1-\boldsymbol{y}^{(t)})\log(1-\boldsymbol{o}^{(t)})+\frac{\lambda}{2}\|\omega\|_2^2 \quad (3)$$

where $\frac{\lambda}{2}\|\omega\|_2^2$ is the item of L^2 regularization to avoid over-fitting. We first calculate the gradient $\nabla_{\boldsymbol{s}^{(t)}}L$ of the last state $\boldsymbol{s}^{(t)}$, and update the weight matrix $\boldsymbol{V}$ based on controlled gradient descent. Then we recursively calculate the gradient of all states from $\boldsymbol{s}^{(t-1)}$ to $\boldsymbol{s}^{(1)}$. The matrix $\boldsymbol{U}$,$\boldsymbol{W}$ are updated during the iteration. This is called Back Propagation Trough Time (BPTT). The green arrows in Fig.2 indicate its order of calculation.

C. Formal Tools

In this research we use the structure-based reliability model to calculate the overall reliability of software. The most popular one is the discrete time Markov chain (DTMC) model [11] which uses diagram similar to workflow to describe the control-transfer relationships between modules. Assuming that module N_i has the reliability degree R_i and the transfer probability $P_{i,j}$, the product $R_iP_{i,j}$ expresses the probability that N_i has been executed successfully and then transferred to N_j. This is the probability of one-step transfer in the Markov chain.

We can get the one-step probability of any pair of modules and form a matrix $\boldsymbol{Q}$ called one-step stochastic transfer matrix. The Neumann series of matrix $\boldsymbol{Q}$ is

$$\boldsymbol{S}=\boldsymbol{I}+\boldsymbol{Q}+\boldsymbol{Q}^2+\ldots=\sum_{k=0}^{\infty}\boldsymbol{Q}^k \quad (4)$$

where $\boldsymbol{I}$ is the identity matrix. Let's set the u^{th} row of $\boldsymbol{S}$ belongs to the starting module N_u and the v^{th} column belongs to the ending module N_v. So the system reliability can be computed as:

$$R_{sys}=S_{u,v}\,R_v \quad (5)$$

which is expressed as the probability of successfully reaching N_v and successfully executing N_v. More details are not restated here. A predictable difficulty is that the parameters R_i and $P_{i,j}$ are not easily known in practical applications. We will focus on this issue in the next section.

Another problem is the expression of the DTMC model. In fact, there are currently no tools to support the automatic creation of such diagram. As the number of modules increases, graphical representations and calculations based on this will become more complex and difficult. Besides using only directed arcs is not enough to represent all relationships between modules in a local structure, such as parallelism. We have proposed an easy-to-use method in our previous studies [20] for this. The basic idea is to use an expression $N_i \oplus N_j$ instead of the arc $N_i \rightarrow N_j$ in diagram. The operator $\oplus$ denotes the most basic relationship between modules—motivating, which leads to the generation of control transfer. The advantage of expression is that it is precise and unambiguous, especially when dealing with more complex and larger-scale situations.

We can introduce more operators to express more complex relationships so that these algebraic operators can form a complete algebraic system. It is formally equivalent to ordinary algebraic expression, which can be automatically parsed by a formal language automata such as LL, LR, SLR, etc.. Using this tool we can automate the calculation of the DTMC model. More details are confined to space and are not described here.

III. Experimental Setup

We first give our research object and explain why. Then the detailed description of experimental data is presented. We use some methods to process the metric data for better performance. Finally, we propose a scheme in order to integrate metrics into reliability degree for one module.

A. Projects and Datasets

In this paper we use two open-source projects–jEdit [21] and Apache Ant [22] for research. Both belong to the development tools series and have the same version length in the PROMISE repository [23]. Table I shows the use of PROMISE data in our approach.

TABLE I. The Training Set and Test Set for RNN

Project	Historical Version (for training)	Current version (for test)
jEdit	3.2, 4.0, 4.1, 4.2	4.3
Ant	1.3, 1.4, 1.5, 1.6	1.7

TABLE II. Part of The Structural Information of jEdit4.3

Package	Description	Files	Classes	Mark
browser	File system browser.	10	10	N_1
bsh	Bean shell.	115	106	N_2
buffer	Buffer event listener.	18	18	N_3
bufferio	I/O request for buffering.	6	6	N_4
bufferset	A set of buffer.	4	4	N_5
gui	GUI controls and dialog boxes.	86	88	N_6
…	… …	…	…	…

Total: **23** packages, **496** files, **492** classes

The two projects are properly sized and representative for development technology. In addition to the metric data, we also need to analyze the structural information. When we carry out reliability engineering at early stage, we consider ourselves as developers and designers. So we can get the necessary information of structure from the design documentation or source codes. This information includes packages, files, and classes. Table II lists part of them in jEdit4.3 for example.

We has also learned that there are **15** modules(packages), **785** files and **745** classes in Ant 1.7. Similarly, we need to understand the structure of earlier versions. It takes a lot of effort, which is why only two open source projects are used as research objects.

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 metric data of one module.

B. Metric Data Processing

There are usually three categories–traditional metrics, OO metrics and process metrics, to classify metrics [24]. Sometimes traditional and OO are called code metrics. The data from PROMISE are summarized at the method-level, class-level and file-level respectively. At the method-level, the number of lines of code (LOC) and the cyclomatic complexity (CC) are still suitable for code analysis inside a class, which existed before object-oriented programming appeared. The CK [25] set has a wide range of applications, but there are also metrics that emphasize perspectives such as encapsulation, coupling, etc. [26]. The eight metrics recommended by Moser et al. [27] have typical process characteristics, and are further improved in the MJ [28] set.

In this paper, metric data are divided into six categories: complexity, coupling, cohesion, inheritance, size and process. Table III lists all relevant metric elements.

TABLE III. Metric Division in This Paper

Category	Metric Element	Mark
complexity	AMC MAX_CC AVG_CC	c_1
coupling	CBO CA CE IC CBM	c_2
cohesion	LCOM LCOM3 CAM	c_3
inheritance	DIT MOA MFA	c_4
size	WMC NOC RFC NPM LOC DAM	c_5
process	NR NDC NML NDPV	c_6

We train different RNNs from the above six aspects to identify defect-prone classes, and use TensorFlow 1.4 to solve this two-classification problem. We first initialize $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ randomly, then set $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{s}^{(0)}$ to 0. We use different data sequences when training different RNNs. As shown in Table IV, the input series consist of four vectors–$\boldsymbol{x}^{(1)}$~$\boldsymbol{x}^{(4)}$, while the vector $\boldsymbol{x}^{(5)}$ is treated as test data. We define the same sequence for all classes in all modules, and use $\boldsymbol{x}^{(5)}$ as

test too. So the size of the training set of jEdit is 492, and Ant is 745. Note that the same model still needs to be trained five more for other categories.

C. Aggregation Scheme

While the metric data we use are only collected at the class-level, it is necessary to propose a descriptive scheme to aggregate them into the value of software system reliability finally.

TABLE IV. AN INPUT DATA SERIES CONSISTING OF VECTORS (E.X. THE BROWSERVIEW CLASS IN JEDIT PROJECT)

Coupling Metric	Version				
	v3.2 $\boldsymbol{x}^{(1)}$	***v4.0*** $\boldsymbol{x}^{(2)}$	***v4.1*** $\boldsymbol{x}^{(3)}$	***v4.2*** $\boldsymbol{x}^{(4)}$	***v4.3*** $\boldsymbol{x}^{(5)}$
CBO	13	18	25	24	28
CA	8	11	14	15	15
CE	10	14	20	16	21
IC	0	1	1	1	1
CBM	0	4	4	4	4

Each RNN classifies all the classes into two types: defeat-prone or reliable. So the total number of defeat-prone classes can be counted in one aspect. For the defeat-prone class (DPC), we give the following definitions to mark the training data series: (i) For a certain version, if a class has bug commit, it is defined as DPC; (ii) If the metric of a class is obviously abnormal, the class is defined as DPC under the category which the metric belongs to (in Table III).

Note that the historical version is only aware of bug reports. The class under the current version, taking the class BrowserView in jEdit4.3 as an example, belongs to the situation under development and does not have a bug report. This is the difficulty of early reliability prediction. The input vector $\boldsymbol{x}^{(5)}=[28\ 15\ 21\ 1\ 4]^{\mathrm{T}}$ only works from the coupling aspect. The BrowserView class still needs the rest from other five aspects to aggregate. We define the reliability influence (*RI*) of each category in one module, which is calculated by

$$RI(c_i) = [1 - (\frac{N_{dpc}(c_i)}{N_{all}})] \ *100\% \qquad (6)$$

where $N_{dpc}(c_i)$ is the number of defeat-prone classes in a module, N_{all} is the number of classes in a module, and c_i indicates at a specified aspect (from c_1-'complexity' to c_6-'process').

Aggregation strategy can significantly alter correlations between software metrics and the defect count. Zhang [17] pointed out that the summation strategy can often achieve the best performance when constructing models predict defect rank or count. In this paper, we use the summation strategy to aggregate the reliability influence of six categories (aspects) of metrics into the reliability of individual modules. It is calculated as:

$$R_{\text{module}} = \sum_i \left(r_i * RI(c_i) \right) \qquad (7)$$

where r_i is weight of the c_i metric category. We think that any category represents a different logic, so let them have the same weight. Here is r_i= 1/6 (i=1~6).

In the actual development process, reliability engineering often needs to be carried out by architects and coders. The module developer may calculate R_{module} by counting the related metrics. And we suggest a formal tool to apply R_{module} into a DTMC model in Section II. So module developers are additionally required to submit related expressions based on their understanding of the workflow. For example, developer of the module browser (recorded as N_1) in jEdit4.3 project should submit: (i) R_1; (ii) $N_1 \oplus N_{21}$. The expression $N_1 \oplus N_{21}$ replaces the directed arc to describe the control transfer flow between N_1 and N_{21}. It indicates that further processing of file system will go to the module utilities (N_{21}).

Module developer can separately submit expressions that confirm design intent, which formally start with the developing module and link all possible next modules in the workflow. In some cases, architects can also submit or modify expressions based on overall understanding. Algebraic expressions are precise and unambiguous, and it is lightweight and easy to use for engineers..

We can collect an expression set finally which implicitly contains two key parameters required for the DTMC model: R_i and $P_{i,j}$. We have already discussed the calculation process of R_i, which is reliability of the ith module in one system. The transfer probability $P_{i,j}$ indicated by $N_i \oplus N_j$, is equally divided by all possible transfers from module N_i. The syntactic parser can automatically parse out this information by scanning the expressions.

IV. RESULTS AND DISCUSSION

The experimental results and corresponding discussion are presented in this section.

A. Results

In this research we focus on the methods of early reliability engineering in practical projects, and solve the following two problems:

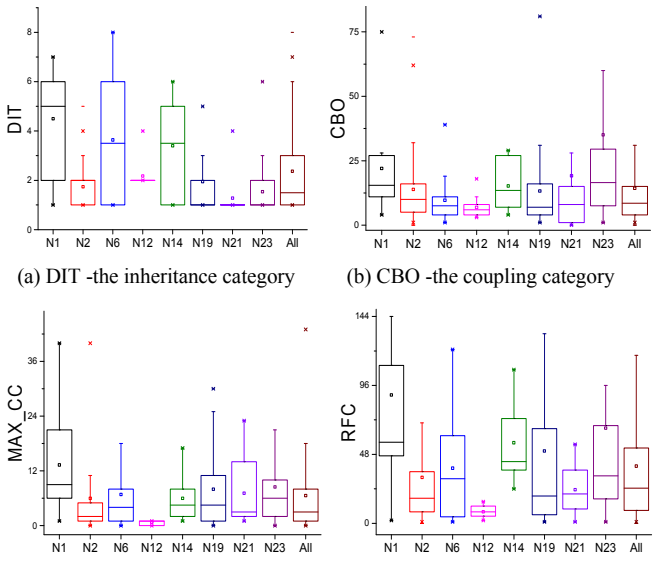
- From the perspective of early reliability analysis, what kind of structure granularity is appropriate and how to use software metrics?
- How to evaluate the overall system reliability in practical engineering?

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 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.

Different packages show significant differences in metrics due to the diversity of functionality and developer. As shown in Fig.3, the distributions of metric data are obviously different, and most of them are different from the whole distribution. It denotes that the coding style of each module is possibly different, that is, the quality of each module in the same software is also different. Besides, the

life cycle and application scope of many packages (or components) goes beyond the project itself.

Therefore, we think that for each different module, the reliability influence from its metric data should be analyzed separately. Table V lists the model classification results for all classes in one package (module). Using (6) and (7), we calculate reliability value of the browser module to be 88.33%. Similarly, the reliability values of the remaining 22 modules can be obtained. The values of 15 modules in project Ant1.7 are also obtained. Table VI lists the



(a) DIT -the inheritance category (b) CBO -the coupling category

(c) MAX_CC -the complexity category (d) RFC -the size category

Fig.3. Boxplots of metrics from different categories. Four metrics and eight modules are both randomly selected.

TABLE V. THE CLASSIFICATION RESULTS OF RNNS (E.X. THE BROWSER PACKAGE IN JEDIT4.3)

Class Name	Classification Based on Six Metric Categories					
	c_1	c_2	c_3	c_4	c_5	c_6
BrowserCommandsMenu	0	0	0	0	0	0
BrowserIORequest	0	0	0	0	0	0
BrowserListener	0	0	0	0	0	0
BrowserView	0	0	1	0	0	0
FileCellRenderer	0	0	0	0	0	0
VFSBrowser	1	0	1	0	0	0
VFSDirectoryEntryTable	0	1	0	0	0	1
VFSDirectoryEntryTableModel	1	0	0	0	0	0
VFSFileChooserDialog	0	0	0	0	0	1
VFSFileNameField	0	0	0	0	0	0

1: defeat-prone, 0: reliable

Then the reliability value of all modules should be integrated with the DTMC model. From the perspective of reliability engineering, we need module developer to provide the relationship between this module and other modules in addition to R_{module}, which can be described conveniently by algebraic expressions. Finally, an expression set can be established to complete the summary calculation instead of the graph. The set of jEdit4.3 is shaped as follows:

$$\left\{\begin{array}{l} N_{12} \oplus N_3, N_{12} \oplus N_6, N_{12} \oplus N_{19}, N_{12} \oplus N_{20}, \\ N_1 \oplus N_{21}, N_2 \oplus N_{21}, N_3 \oplus N_5, N_4 \oplus N_5, \\ \cdots, \\ N_{20} \oplus N_{23}, N_{21} \oplus N_{23}, N_{22} \oplus N_{23}, \\ N_{23} \oplus N_3, N_{23} \oplus N_{24} \end{array}\right\} \quad (8)$$

The starting module is N_{12} (msg) in a workflow, which means that the message occurs as the start of the business. In order to facilitate model calculation, we constructed the termination module N_{24} (R_{24}=100%), and considered that the business termination requirement is initiated only by the module N_{23} (core).

We also establish the algebraic expression set for another two versions of jEdit and three versions of Ant. Notice that the convention of current version has been expanded here. For jEdit when setting 4.3 to the current version to be evaluated, the required input series is 3.2→4.0→4.1→4.2. The time length is 4. It should change into 3.2→4.0→4.1 when the current version is set to 4.2, and the length is 3. In the extreme case of the current version 4.1, its input series length is only 2. The same situation exists in the project Ant.

We use different time series lengths for comparison. And two traditional reliability models—classical G-O model [29] and Huang's model [30] will be used for performance comparison too. They belong to the growth model, the former is representative of the classical model, and the latter has excellent predictive performance because of the integration of testing effort. These two models use failure data at period of testing, which are later than the structured model used in this paper. That is, these models cannot be used to guide design or development.

First, we calculated the reliability values for each module in three different current versions. Limited to space, part of the two-item module results are shown in Table VI.

TABLE VI. PART OF THE MODULE'S CALCULATION RESULTS IN TWO OO PROJECTS

Module	jEdit			Module	Ant		
	v4.1	*v4.2*	*v4.3*		*v1.5*	*v1.6*	*v1.7*
browser	86.39	87.80	88.75	dispatch	–	–	96.87
bsh	88.41	91.84	94.38	filters	92.35	95.64	97.02
buffer	87.45	89.33	95.81	helper	89.12	91.08	94.14
bufferio	–	–	93.54	input	91.56	93.25	96.32
bufferset	–	–	94.11	launch	–	91.39	92.61
gui	85.23	89.32	91.26	listener	96.97	88.55	92.74
…	…	…	…	…	…	…	…

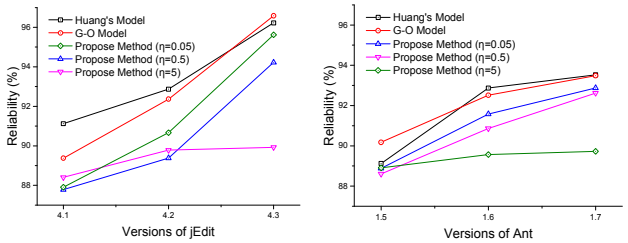
The underlined items in the table describe the special circumstances that may be encountered during the calculation. E.g:

a) The v4.3 value of the modules bufferio & bufferset in jEdit. These two modules are both newly added. In fact, they are derived from the module buffer in previous versions. The

simplification of one module greatly increase the reliability value of the module buffer. This is in line with the original design. The value of the two new modules can only be estimated simply by not having a time series. The estimation rules refer to the definition of *dpc* in Section III.C

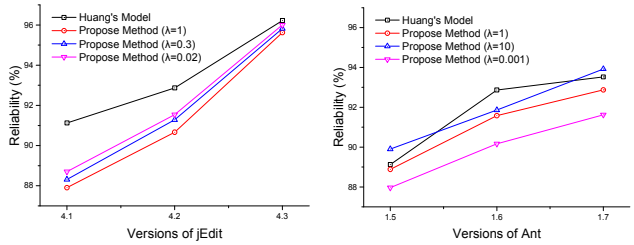
b) The v1.7 value of the module dispatch in Ant. It belongs to the real new function module in the last version. Its value also need to be estimated without using the RNNs.

c) The v1.6 value of the module launch in Ant. In version 1.6, it belongs to the new functional module. But we still want to use RNNs in version 1.7. For this case, we construct a sequence of full length with the same value.



(a) jEdit (b) Ant

Fig.4. The reliability trend curves on two projects by using different evaluation methods. The initial learning rate η is set to 5, then adjusted to 0.5 and 0.05.



(a) jEdit (b) Ant

Fig.5 The curves of different values of regularization parameter λ in two projects when the learning rate is set to 0.05. In jEdit, we experimented λ with a set of values—1, 0.3, 0.02. Then in Ant, we tried another values—1, 10, 0.001 which have a larger span.

Then through the algebra tool, we calculated the overall reliability of three versions for the two OO projects. As can be seen from Fig.4, the reliability trends from version 4.1 to 4.3 in jEdit and from 1.5 to 1.7 in Ant conform to the results of the G-O or Huang's model. And for the main target jEdit4.3 and Ant1.7, the evaluation value of this paper is very close to the tranditional reliability models. Note that it is not possible to use actual test failure data in early evaluation of reliability in contrast to SRGMs like G-O and Huang's. It means that reliability engineering could be implemented early in the development process if the software metrics were used properly.

We set the initial learning rate to 5 in both projects. And we also gradually experimented with two smaller values—0.5&0.05. It is found that the result does not converge to a better value when the set value is 5. In contrast, smaller learning rates have better performance in small-scale dataset, just as the metric data used in this article. In Fig.5, we experimented different values of another hyperparameter—λ, and the results show that the optimal value for the regularization parameter in this research is at 10^{-2} magnitude.

B. Discussion

Unlike SRGMs, the early reliability model can only be based on analysis of software structure. As we mentioned in the first section, the modeling data is limited. Therefore, it is meaningless to compare the predictive performance with a class of models based on failure data. Most traditional reliability models, such as SRGMs, are utilized to optimize the release strategy of software, i.e. 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.

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 II, 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 solves the difficulty of applying the structural reliability model to practice.

The numerical values evaluated by the proposed method are lower than the traditional methods'. 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 believe that the reliability curve increases with bugs fixing. Their evaluations are relatively optimistic in general.

The threat to our treatment mainly arises from applicability of our method on other OO projects. The metric data can be obtained from public libraries , but also directly calculated from source codes. And structural information can be analyzed from design documents and source codes. We believe that the applicability of proposed method is not a problem. Furthermore, it is recommended that the non-java projects should be tested using our approach and the results may be slightly different.

The metrics and its classification we used are popular and commonly investigated in defect prediction literature. In the same kind of metrics, correlation analysis technique is used to eliminate redundant impacts. The study on more correlation analysis techniques is required. In addition, the DTMC model used in this paper is the most important structure analysis model at present. The proposed framework uses formal techniques to solve 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 model, such as CTMC, to utilize this framework may prove fruitful.

V. Conclusions

The main work of this paper is to provide a complete solution for early reliability evaluation used in actual software project. We use the RNN to process metric data from all classes to identify which is defeat-prone in the current version. Then we propose a method for aggregating the RNNs' results into module reliability. We present a framework which can automatically calculates the overall reliability value based on the introduced formal tools. Through using a series of technologies, our research solves the problem of how to implement structural reliability model in practical projects. Evaluation on two OO projects shows that it can achieve an approximating result vs. traditional models which are based on software failure data. This

research provides new ideas and methods for the empirical research of reliability.

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, pp. 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] 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.

[6] H. Sukhwani, J. Alonso, K. S. Trivedi, and I. Mcginnis. "Software reliability analysis of nasa space flight software: A practical experience," *IEEE International Conference on Software Quality, Reliability and Security*, vol. 3, pp. 386-397, 2016.

[7] L. Aversano, and M. Tortorella. "Analysing the reliability of Open Source software projects," *in Proc. 10th International Joint Conference on Software Technologies*, pp. 348-357, 2016.

[8] K. Honda, N. Nakamura, H. Washizaki, and Y. Fukazawa. "Case study: Project management using cross project software reliability growth model," *IEEE International Conference on Software Quality, Reliability and Security Companion*, pp. 41-44, 2016.

[9] 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.

[10] B. Littlewood. "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, vol. R-28, no. 3, pp. 241-246, 1979.

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

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

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

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

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

[16] 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.

[17] 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 Transactions on Software Engineering*, vol. 43, no. 5, pp. 476-491, 2017.

[18] L. Fiondella, A. Nikora, and T. Wandji. "Software reliability and security: challenges and crosscutting themes," *IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 55-56, 2016.

[19] 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.

[20] 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.

[21] http://www.jedit.org

[22] http://ant.apache.org

[23] http://openscience.us/repo/index.html

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

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

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

[27] R. Moser, W. Pedrycz, and G. Succi. "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," *ACM/IEEE, International Conference on Software Engineering*, pp. 181-190, 2008.

[28] 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.

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

[30] 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.