The Journal of Systems and Software 101 (2015) 236–244

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Improving software reliability prediction through multi-criteria based dynamic model selection and combination

Jinhee Park*, Jongmoon Baik

Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Guseong-dong, Yuseong-gu, Daejeon, Republic of Korea

ARTICLE INFO

Article history:
Received 6 February 2014
Revised 22 September 2014
Accepted 12 December 2014
Available online 16 December 2014

Keywords:
Software reliability prediction
Decision trees
Multi criteria

ABSTRACT

In spite of much research efforts to develop software reliability models, there is no single model which is appropriate in all circumstances. Accordingly, some recent studies on software reliability have attempted to use existing models more effectively in practice (e.g., model selection and combination). However, it is not easy to identify which model is likely to make the most trustworthy predictions and to assign appropriate weights to models for the combination. The improper model selection or weight assignment often causes unsuccessful software reliability prediction in practice, which leads to cost/schedule overrun. In this paper, we propose a systematic reliability prediction framework which dynamically selects and combines multiple software reliability models based on the decision trees learning of multi-criteria. For the model selection, the proposed approach uses the empirical patterns of multi-criteria derived from models. Reduced error pruning decision tree identifies the models with the best predictive patterns and automatically assign a weight to each model. Then, the identified models fall into two groups according to the likelihood of over- or under-prediction, and the competitive models from each group are combined based on their given weights. From the evaluation results, our approach outperformed existing methods on average prediction accuracy.

1. Introduction

Since unreliable software may cause critical damages to humans or business reputation, software practitioners have made significant efforts to achieve high software reliability during testing process. In this circumstance, their main interest is to control the software testing process by predicting when a failure rate would have fallen below an acceptable threshold. The success of this activity highly depends on the prediction accuracy of software reliability models. For the better reliability prediction, numerous software reliability growth models (SRGMs) have been proposed over the past four decades. As a result, software practitioners now have abundant SRGMs, while none of which work optimally across projects (Raj Kiran and Ravi, 2008).

The practical use of SRGMs has been limited for many reasons. Since project data varies considerably and often does not comply with the underlying assumptions of models, practitioners have no reliable way of determining the model in advance (Stringfellow and Andrews, 2002). Accordingly, they mostly compare the performance of models on observed data in various aspects using multiple criteria and select one. However, models shows the different performances according to adopted criteria, and relative priority of models in each criterion may change as testing proceeds. Furthermore, it is widely known in the software reliability engineering domain that the goodness-of-fit (GOF) performance to the observed data does not directly link to the prediction performance (Xiao and Dohi, 2013). In this circumstance, practitioners have difficulties in identifying which model is likely to make the most trustworthy predictions.

The existing works on software reliability prediction have focused on developing more detailed and potentially more complicated models instead of using existing models more effectively in practice. Although some methods use multi-criteria information to find an optimal model, they do not focus on the prediction performance of models, which is a major concern for practitioners. According to ABDEL-GHALY et al. (1986), since only meaningful issue for a user is whether future failure behavior can be accurately predicted from a model, it is required to understand the empirical relationship between the relative performance of models on observed data and their prediction performance. Another method combines the abilities of each model after applying all candidate models, but the improper weight assignment for each model often causes unsuccessful software reliability prediction in practice.

This paper suggests a systematic approach for making more accurate and robust software reliability prediction by decision tree learning, and a tool to support the approach. This approach first searches the informative criteria for the prediction of the specified range. The joint use of these criteria gives weights to software reliability models according to the likelihood of better prediction. Using these weights, we dynamically identify the models that are more likely to make

* Corresponding author. Tel.: +82 42 350 7756; fax: +82 42 350 7859.
E-mail address: jh_park@kaist.ac.kr (J. Park).

http://dx.doi.org/10.1016/j.jss.2014.12.029
0164-1212/

the best prediction. To further improve the predictive capability of the approach, the identified models are combined according to their tendencies of over- and under-prediction.

In summary, major contributions of this work are as follows: (1) the proposed approach enables more accurate and robust software reliability prediction by selecting and combining existing models effectively, which is a major issue for practitioners. (2) The proposed technique is also automated, and this can mitigate the generalization problem of an individual SRGM. (3) This paper contains the extensive empirical results of various software reliability prediction techniques with thirty software reliability datasets.

The paper is structured as follows. The motivating problems and related work on software reliability prediction are described in Section 2. In Section 3, we describe the comparison criteria used in this paper. Section 4 explains our decision tree based software reliability prediction framework, and the experimental design for the approach is presented in Section 5. The experimental results and the limitations of the work are discussed in Section 6. Finally, Section 7 concludes this paper.

2. Motivating problem and related work

This section is separated into two parts. The first part explains the motivating problem of this paper. The second part describes the related work on software reliability models, model selection, and combination.

2.1. Motivating problem

In spite of the proliferation of SRGMs, it has been known that there is no single model appropriate for all cases (Amin, Grunske, & Colman, 2013; Raj Kiran and Ravi, 2008). Accordingly, it is necessary to consider multiple candidate SRGMs for a given project. However, it is often difficult to know which models to be applied in advance because SRGMs have different underlying assumptions that are often violated in practice (e.g., perfect debugging, and immediate correction of faults, etc.). Current practice is to apply several SRGMs and select the best one based on comparison criteria (Stringfellow and Andrews, 2002), but this is a complicate task usually guided by the experience of reliability experts. For practitioners, it is more important to accurately predict the future behavior of software rather than to merely explain past behavior (Khoshgoftaar and Woodcock, 1991; Konishi and Kitagawa, 2008). Therefore, it is necessary to develop the method with which the practitioners can easily adopt and obtain high reliable prediction results.

After fitting models to observed data, practitioners can evaluate the performance of models in various aspects. For example, mean square error (MSE) evaluates the overall fitness of the curve, while Bias sees the tendency to lean to one side. Predictive ratio risk (PRR) and Noise measures the underestimation risk of models and the smoothness of the estimated values respectively. Although it is preferable that there exists the model that provides good performance in all criteria, practitioners mostly face conflicting criteria (e.g., MSE: best, Bias: worst, Noise: worst), and there is no research on the joint effects of those criteria on the short-term and long-term prediction performance of models. In particular, at the unstable stage of testing (e.g., 40% or 50% of planned testing), it is difficult to know what characteristics of models on the observed data provide most trustworthy short-term or long-term predictions. To recognize those characteristics, it is required to analyze the empirical relationship between multiple criteria on the observed data and prediction performance at different testing periods. Furthermore, although we can find the best one among candidate SRGMs, the accuracy of software reliability prediction is limited to the predictive capability of the selected SRGM. Since models often over-predict or under-predict the actual data, if the tendency of under-or over-prediction of models can be identified, it is possible to make a better software reliability prediction by combining models.

2.2. Related work

Various types of SRGMs have been developed and used in many different industry sectors since the 1970s (Goel, 1985). In particular, non-homogeneous Poisson process (NHPP) based models have been widely applied to predict the general trend of software failure processes during the testing phase (Andersson, 2006; Wood, 1997). The NHPP based SRGMs are determined by the assumptions of software fault detection and removal processes. The detailed information of these assumptions is described in Kharchenko et al. (2002). To relax some impractical assumptions of models, many modifications of SRGMs have been proposed by using different fault content and fault detection rate functions (Huang et al., 2003; Nordmann, 1999; Pham, 2006; Zhang et al., 2003). Table 3 in Section 5 describes NHPP based SRGMs with different types of functions.

To solve the generalization problem of single SRGM, some studies on software reliability have focused on a model selection problem (Andersson, 2006; Goel, 1985; Sharma et al., 2010). Goel (1985) noted that an optimal software reliability model can be selected based on the underlying assumptions of candidate models. If a testing process corresponds closely with the assumptions of some models, the models can be applied for reliability estimation. Kharchenko et al. (2002) proposed an assumption matrix based model selection technique. However, many assumptions of models are often violated in practice, and the selected models by the assumptions do not guarantee the accuracy of reliability prediction for a project (Dharmasena et al., 2011).

As a result, some studies have proposed methods to select the best model after applying all candidate models to the observed data (Lyu and Nikora, 1992; Sharma et al., 2010; Stringfellow and Andrews, 2002; Wood, 1996). To select the model, these methods use multi-criteria information such as the sum of the rankings of all comparison criteria or the distances from the optimal values of all possible criteria. However, the selected models by these methods are often unsuccessful in the software reliability prediction because they do not focus on the prediction performance of models. Furthermore, such model selection methods do not work well if no model is appropriate for predicting particular trends in software failure data, i.e., the software reliability prediction is limited to the capabilities of the selected models.

As an alternative to model selection, the combination of models has been studied. Lyu and Nikora (1992) combined all candidate models based on assigned weights to reduce the risks that can be caused by depending on any particular model. While this approach simply combines the results of models based on assigned weights, Su and Huang (2007) built the dynamic weighted combinational model (DWCM) based on a neural network to predict software reliability. However, this approach focuses on the fitness to the observed data, and it may cause the over-fitting problem (Yang et al., 2010).

Due to the applicability across different software projects, data-driven software reliability models (DDSRMs) have been developed. These are usually time series models (e.g., traditional autoregressive integrated moving average models and various machine learning technique based models), and time lag terms for software failure data are used as model inputs. Since artificial neural networks have been first applied for software reliability prediction (Karunanithi, 1992; Karunanithi et al., 1992), many DDSRMs with different machine learning techniques have been proposed to improve software reliability prediction accuracy (Cai et al., 2001; Hu et al., 2007; Pai and Hong, 2006; Tian and Noore, 2005a; Tian and Noore, 2005b). Among them, support vector machines (SVMs) have been successfully employed for building DDSRMs (Moura et al., 2011). Some recent studies show that the support vector machine for regression (SVR) model outperformed

other techniques such as autoregressive integrated moving average models, general regression neural networks infinite impulse response locally recurrent neural network, and radial basis function neural network models in the software reliability prediction (Kumar and Singh, 2012; Moura et al., 2011; Pai and Hong, 2005). However, since these approaches have focused on the next-step prediction, their predictive capabilities in the long-term are questionable, and more research efforts on the performance of DDSRMs are required.

Our approach can be viewed as a hybrid approach where empirically selects and combines existing SRGMs, rather than developing more detailed models. The strategy is to find the empirical patterns of comparison criteria derived from the models that have given relatively good predictions. There are various criteria focusing on different aspects of the model performance. Such criteria values are available after fitting models. By identifying the joint effects of these criteria on prediction, we can understand the general characteristics of the models that show the best predictive accuracy and can assign weights to models according to how likely each model is to make the best prediction. It is possible to use those characteristics for model selection. To explore the possibility of further improvement in prediction accuracy, we can also examine the various combinations of selected models with their assigned weights.

3. Comparison criteria for software reliability models

This section lists the criteria widely used in the software reliability domain. Various criteria have been used to compare the models in the reliability engineering domain. Each comparison criterion focuses on the different aspect of the model performance. In this study, we employ the most of criteria used in Sharma et al. (2010) except some highly correlated criteria. All criteria below share the same notations. m_i is the total cumulated number of failures observed within time $(0, t_i)$, $\hat{m}(t_i)$ is the estimated cumulative number of failures up to the time t_i, k represents the sample size at the time of model fitting, $i = k, k+1, \ldots, n$. n is the sample size at the termination of testing, and p denotes the number of parameters.

Criterion 1. MSE gives the measure of the overall differences between the estimated values and the actual observations (Zhang and Pham, 2006). The smaller MSE indicates a small fitting error. This criterion is calculated as follows:

$$\text{MSE} = \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2}{k-p} \tag{1}$$

Criterion 2. Mean absolute error (MAE) measures the absolute errors unlike MSE (Chiu et al., 2008). This criterion is less sensitive than MSE to the occasional large error and is calculated as follows:

$$\text{MAE} = \frac{\sum_{i=1}^{k} |m_i - \hat{m}(t_i)|}{k-p} \tag{2}$$

Criterion 3. Rsquare measures how successful the fit is in explaining the variation of the data (Chiu et al., 2008). This criterion is calculated as follows:

$$\text{Rsquare} = 1 - \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2}{\sum_{i=1}^{k} \left(m_i - \sum_{j=1}^{k} m_j/k\right)^2} \tag{3}$$

Criterion 4. Noise measures the smoothness of predicted values (Huang et al., 2007). This criterion is calculated as follows:

$$\text{Noise} = \sum_{i=1}^{k} \left| \frac{\lambda(t_i) - \lambda(t_{i-1})}{\lambda(t_{i-1})} \right| \tag{4}$$

Criterion 5. Bias is the average of errors between the estimated values and the observed actual data (Pillai and Nair, 1997). The bias closer to zero indicates the unbiased estimation. This criterion is calculated as follows:

$$\text{Bias} = \frac{\sum_{i=1}^{k} (\hat{m}(t_i) - m_i)}{k} \tag{5}$$

Criterion 6. Variation is the standard deviation of Bias and calculated as follows (Pillai and Nair, 1997):

$$\text{Variation} = \sqrt{\frac{1}{k-1} \sum_{i=1}^{k} (m_i - \hat{m}(t_i) - \text{Bias})^2} \tag{6}$$

Criterion 7. PRR measures the risk of underestimation. It imposes a higher penalty to the model that has underestimated the number of failures at any given time (Zhang and Pham, 2006). This criterion is calculated as follows:

$$\text{PRR} = \sum_{i=1}^{k} \left(\frac{(\hat{m}(t_i) - m_i)}{\hat{m}(t_i)} \right)^2 \tag{7}$$

Criterion 8. Weighted least square error (WLSE) gives more penalties to the recent errors. This criterion is obtained from the estimation technique by Li and Malaiya (1993). The smaller WLSE indicates the better fitting to recent data. This criterion is calculated as follows:

$$\text{WLSE} = \sum_{i=1}^{k} (\hat{m}(t_i) - m_i)^2 \times (c + e \times i),$$

$$\sum_{i=1}^{k} (c + e \times i) = k, \quad 0 \le c \le 1 \tag{8}$$

where c represents the degree of difference in weights, and e is determined after setting c value.

Criterion 9. End-point prediction reflects the difference between the predicted number of failures and the actual number of all occurred failures until the end of test time t_q (Su and Huang, 2007). The observed failure data up to t_e ($t_e \le t_q$) are used to estimate the parameters of models. This is end-point prediction (EP) and calculated as follows:

$$\text{EP} = |m_q - \hat{m}_e(t_q)| \tag{9}$$

Criterion 10. Mean error of prediction (MEOP) is a criterion to evaluate the overall quality of prediction (Zhao and Xie, 1992). This criterion is defined as follows:

$$\text{MEOP} = \frac{\sum_{i=k}^{n} |m_i - \hat{m}(t_i)|}{n-k+1} \tag{10}$$

4. Decision tree based software reliability prediction framework

This section presents the approach that predicts software reliability through multi-criteria based model selection and combination. The primary goal of this approach is to make more accurate software reliability predictions based on the predictive learning of multi-criteria. Fig. 1 shows the overall procedure. This approach first trains the empirical patterns of criteria by decision tree algorithms. The trained decision tree model will generally include only a subset of the criteria that are deemed truly informative for the prediction. The approach automatically identifies the models that are likely to make the trustworthy predictions depending on empirical evidences, rather than the subjective decision of practitioners. To identify the model, reduced error pruning tree (REPTree) (Tapio Elomaa, 2001) learns the predictive patterns of models from the empirical datasets. Then, we classify the tendencies of over- and under- prediction of the identified models based on alternating decision tree (ADTree) (Holmes et al., 2002). The models are combined for better reliability prediction.

4.1. Build decision trees

The proposed approach characterizes the models on software testing data by two decision tree learning algorithms. REPTree trains the empirical criteria patterns that affect the prediction accuracies of models, and ADTree trains the over- and under-prediction patterns,

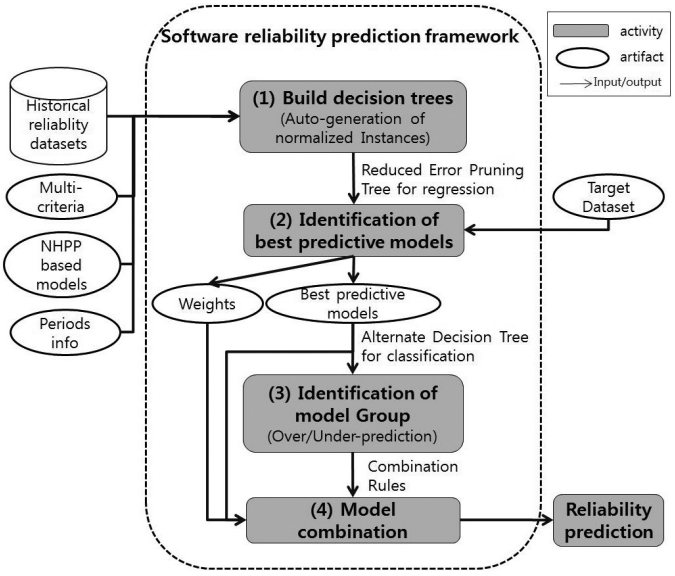

Fig. 1. Overall procedure for a reliability prediction framework.

which is a binary classification problem. These tree learning algorithms visualize the joint effects of criteria and help to find relatively important criteria on each purpose.

Before generating instances of these algorithms, it is required to receive the user inputs that are related to a current and prediction point (%) against the planed testing time. Practitioners apply the models to the observed data at any point against planed testing and make a prediction for the future periods. Then, they decide whether to stop testing or not. The approach partitions each empirical dataset into training and test subsets according to user inputs. Once each example dataset is divided into training and test subsets, the model parameters are estimated on the training subset by a non-linear least squares (NLS) regression which is a well-established statistical technique (Venables and Ripley, 1994). This method is widely used in reliability modeling research (Jones, 1991; Stringfellow and Andrews, 2002; Wood, 1996). After applying the multiple models to the training subset (i.e., observed failure data), the comparison criteria (explanatory variables) are calculated on the training subset, while the predictive criterion (target variable) is calculated on the test subset. The approach applies this calculation to all empirical datasets.

To generate instances for the REPTree algorithm, the proposed approach uses MEOP or EP criterion as a target variable. These criteria represent the prediction accuracies of models. As explanatory variables, we use the ten criteria identified in Section 3. These comparison criteria focus on the different aspect of the model performance. To build ADTree, the proposed approach uses Bias on the test subset as a target variable and some comparison criteria related to over- (or under-)prediction as explanatory variables. Bias measures the tendency of how the prediction leans toward one side.

Before training these instances, a set of measured values of the criteria is normalized in a dataset. The min–max normalization is used to scale numeric values in the range [0, 1], and the following equation can be used (Singh et al., 2010).

$$\bar{d}_i = \frac{\max(d) - d_i}{\max(d) - \min(d)} \tag{11}$$

where $i = 1, 2, 3, \ldots, n$ and d is the set of data values. d_i is the value for normalization, and $\bar{d}_i$ is a result value after normalization. The $\max(d)$ and $\min(d)$ mean the maximum and minimum values respectively in data. In the approach, the normalization is modified to assign a value close to one to the models with better accuracy. In other words, the model with the highest ranking for a specific criterion in a data set is given the value of number one, and the model with the lowest ranking is given the value of number zero. Accordingly, the values between zero and one are assigned to comparison criteria for models.

4.2. Identification of the best predictive models

To identify the prediction performance of models, we adopt a regression strategy. REPTree handles the numeric attributes and implements a regression by pruning a tree built by information gain or variance. The training tuples for REPTree have the form of INSTANCE x [MSE, MAE, Rsquare, Noise, Bias, Variation, PRR, WLSE, CP, CMEOP, MEOP(EP)]. The last criterion is a target variable on the test subset, and the other criteria are predictors on the training subset. Here, current prediction (CP) and current MEOP (CMEOP) indicate the EP and MEOP on the training subset. Since the practitioners cannot measure this criterion at the time of model selection, these are calculated from the model built at the past point in this approach. Each model fitted to a dataset has one training instance. If we apply thirteen models to a dataset, thirteen instances are generated for each dataset.

By using all instances from the empirical datasets, the patterns of these competitive criteria values are used to assess how likely a model is to predict the future failure process of software. After learning those patterns, REPTree scores each model on a target dataset. In other words, REPTree generates the expected MEOP(EP) values of models by using normalized criteria information on the observed data. Those MEOP(EP) values from the REPTree learner are used as model weights, and, the best three models are selected as candidate models. We set the number of candidate models for a combination as three, with a reference to Su and Huang (2007), and those models are combined by combination rules in step 4. The rationale of step 2 is that we can find the better predictive models by considering the joint effects of multi-criteria.

4.3. Identification of model group

The proposed approach classifies the SRGMs into two groups: (1) over-prediction group (plus sign in Bias on the test subset), (2) under-prediction group (minus sign in Bias on the test subset). To help this classification, we introduce a tendency of over- or under-prediction (TOUP) criterion based on the exponential smoothing method as follows:

$$\text{TOUP} = \sum_{i=0}^{m} \alpha \times (1-\alpha)^i \times [error(t-i) - error(t-i-1)] \tag{12}$$

where t is the time of model fitting and $i = 0, 1, \ldots, m$. m is the time for recent observations, $error(t)$ is the deviation between the estimated values and the actual observations at time t, and α is a smoothing constant between 0 and 1. This criterion provides an exponentially weighted moving average of error changes (i.e., the recent changes of fitting errors). In our observation, these change patterns were one of good indicators to decide whether the model would over-predict or under-predict. If the fitting errors of a model are moving to the positive (or negative) direction, the model is expected to over-predict (or under-predict) future software failures. The class-labeled training tuples for a classification have the form of INSTANCE x [Noise, Bias, PRR, TOUP, Group] where the first four criteria are numerical variables and the last criterion is a categorical explanatory variable. For the classification, Bias and TOUP are not normalized to maintain a minus sign.

By using all instances from the empirical datasets, we make classifiers for a model classification. ADTree classifies the models into one of the groups using normalized criteria information on the observed data. ADTree generalizes decision trees and has connections to boosting. This uses a LogitBoost procedure as the underlying boosting algorithm for classification. Holmes et al. (2002) describe the details on

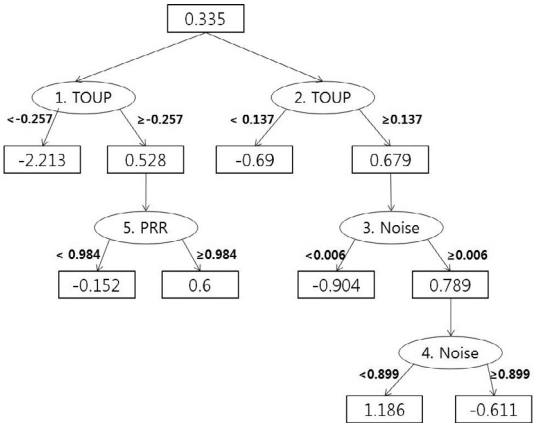

Fig. 2. ADTree model constructed from empirical reliability data sets.

Table 1
Scoring method of ADTree.

Decision node	Classifiers (criteria(values) ≥ threshold)	Score
Initial	–	0.335
Node 1	1. TOUP (0.95) ≥ -0.257	0.528
Node 5	5. PRR (0.7) < 0.984	−0.152
Node 2	2. TOUP(0.95) ≥ 0.137	0.679
Node 3	3. Noise (0.2) ≥ 0.006	0.789
Node 4	4. Noise (0.2) < 0.899	1.186
Final score (sum)	–	3.365

the algorithm. Fig. 2 shows one example of the ADTree that is constructed by using twenty nine datasets among the datasets described in Table 2. It consists of five decision nodes (oval-shaped) as the weak classifiers and eleven prediction nodes (square-shaped) which assign a weight for each class. Each path in the tree is series of weak hypotheses, and sets of nodes on the same path are interpreted as having a joint effect. The final score is the sum of all prediction nodes through which it passes while traversing the tree, and it represents the model group.

For example, if a model has a single INSTANCE x [Noise(0.2), Bias(0.1), PRR(0.7), TOUP(0.95)] for a specific data set, the instance is scored as shown in Table 1. From the root of a tree, we check the conditions in the decision nodes, and find the sum of all prediction nodes through which it passes. Finally, the model is classified as an over-prediction group (i.e., higher score than zero: 3.365) for the dataset. The rationale of this step is that we can identify the characteristics of the models which over-predict or under-predict the data by the combination of multi-criteria.

4.4. Model combination

After dividing the identified models into the over- or under-prediction group, the best single models in both groups are combined based on the given weights. The proposed approach uses an ensemble technique (i.e., linear ensemble based on weighted mean) for the model combination. The ensemble technique produces the predicted number of cumulative failures based on the weighted average of the models with the highest weight in the each of the over- and under-prediction groups. This technique can provide reasonable results because REPTree assigned more weights to the models that are likely to make the better prediction. Moreover, the combination of several single models helps to improve prediction accuracy according to Makridakis and Andersen (1982). Three cases shown in Fig. 3 are examined to make better reliability prediction.

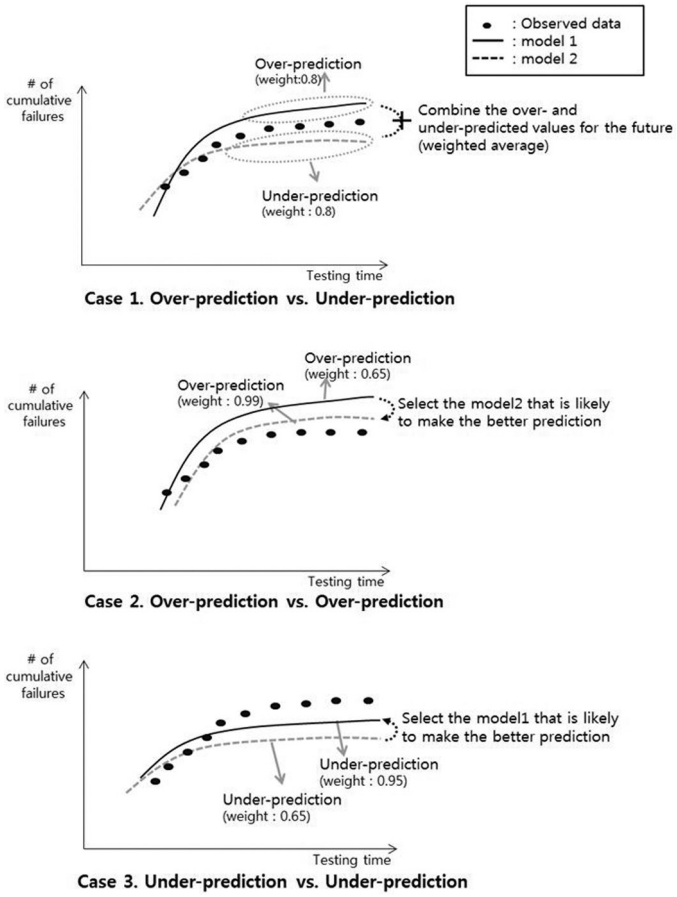

Fig. 3. Three cases on the over- and under-prediction of models.

For the Case 1 which includes the best predictive models in both groups, over- and under-prediction groups, we make reliability prediction by weighted mean of competitive models in both groups. For the Case 2 and Case 3 which mean that no weight is given in one of both groups, we make reliability prediction using the model that has the highest weight in the over- or under-prediction group (i.e., the highest possibility of better reliability prediction).

5. Experimental design

This section describes the experimental design for evaluating the proposed approach. The overall experimental process consists of the three steps as shown in Fig. 4. In Step 1, we analyze the results under the leave-one-out cross validation technique using n historical datasets (Zhou and Leung, 2007). This technique enables to validate the predictive performance of the proposed approach on new unseen datasets. This technique uses one dataset as the validation set and the remaining n - 1 datasets as the training set. After making classifiers from the training set, these classifiers are used on the validation set for evaluating the predictive performance in Step 2. This is repeated until all datasets are validated at least once. The experimental results are analyzed based on three prediction metrics in Step 3.

5.1. Preparing experimental datasets and models

In this experiment, we used the thirty software reliability datasets as study subjects. These datasets were found from various sources (i.e., Cyber Security & Information Systems Information Analysis Center and existing literature on software reliability). They include failure information for the testing process of a particular software development project. They have been often used as benchmark for the

Table 2
Reliability data sets used in this study.

Data set	Application	# of failures	Code (reference)
DS1	Real time command and control	136	CS_1 (Musa et al., 1987)
DS2		54	CS_2 (Musa et al., 1987)
DS3		38	CS_3 (Musa et al., 1987)
DS4		53	CS_4 (Musa et al., 1987)
DS5	Electronic System	461	L_Dataset3 (Lyu, 1996)
DS6		211	L_Dataset4 (Lyu, 1996)
DS7	Aerospace system	133	L_J1 (Lyu, 1996)
DS8		224	L_J2 (Lyu, 1996)
DS9		351	L_J3 (Lyu, 1996)
DS10		188	L_J4 (Lyu, 1996)
DS11		367	L_J5 (Lyu, 1996)
DS12	Commercial software	73	CS_6 (Musa et al., 1987)
DS13		100	L_Tandem r1 (Wood, 1996)
DS14		120	L_Tandem r2 (Wood, 1996)
DS15		61	L_Tandem r3 (Wood, 1996)
DS16		42	L_Tandem r4 (Wood, 1996)
DS17	Military software	38	CS_17 (Musa et al., 1987)
DS18		41	CS_27 (Musa et al., 1987)
DS19		101	CS_40 (Musa et al., 1987)
DS20	Wireless network system	22	L_WNP 1 (Jeske et al., 2005)
DS21		181	L_WNP 2 (Jeske et al., 2005)
DS22	Medical system	231	L_MRS r1 (Stringfellow and Andrews, 2002)
DS23		245	L_MRS r2 (Stringfellow and Andrews, 2002)
DS24		83	L_MRS r3 (Stringfellow and Andrews, 2002)
DS25	Real time control	535	L_Data7 (Tohma and Jacoby, 1989)
DS26		198	L_Data10 (Tohma and Jacoby, 1989)
DS27	Database application	328	L_PL1 (Lo and Huang, 2006)
DS28	Monitoring and real time control	481	L_Tohma 1 (Tohma and Jacoby, 1989)
DS29		266	L_Tohma 2 (Tohma and Jacoby, 1989)
DS30		86	L_Tohma 3 (Tohma and Jacoby, 1989)

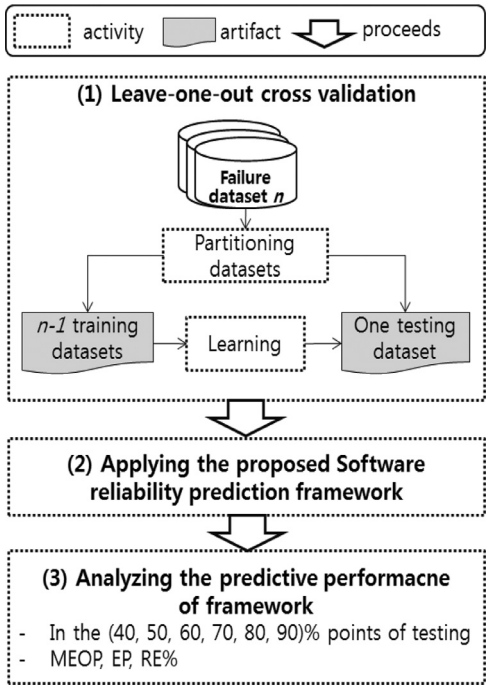

Fig. 4. The overall process of the experiment.

comparison of software reliability models. These datasets are from the various types of applications including real time command and control systems, operating systems, commercial systems, and military applications. The data were mostly collected by careful controls during a system test (Jeske et al., 2005; Li et al., 2011; Lyu, 1996; Stringfellow and Andrews, 2002; Tohma and Jacoby, 1989; Wood, 1996). Table 2 gives a brief summary of all software failure datasets used in this experiment. It also describes the application of a target project based on the project information from each data source, and shows the total number of software failures.

Thirteen NHPP based models as described in Table 3 have been used in this experiment. The form of each model is determined by the combination of the different time dependent fault content and fault detection rate function. We obtain the fitted model by substituting the estimated parameter values into the parameters of SRGMs.

From thirty data sets and thirteen models, we can get more than three hundred instances for training a decision tree because each dataset can generate the instances as much as the number of models. For example, since we use the thirteen software reliability growth models as candidate models, we can have the 390($=13 \times 30$) instances at most for training the adopted decision trees.

5.2. Evaluation criteria

To evaluate the prediction performance of methods, we calculate relative error (RE) as well as EP (Criterion 9) and MEOP (Criterion 10). The following equation for RE can be used (Tian and Noore, 2005a):

$$\mathrm{RE} = \left| \frac{(\hat{x}_i - x_i)}{x_i} \right| \tag{13}$$

where x_i is the actual value of ith data record, and $\hat{x}_i$ represents the model output of ith data record. In this experiment, the portion of RE values within a threshold (5%) is used. Based on these criteria, the outputs (extrapolation) by the approaches are compared with the future software failure data at each partition point (i.e., 40%, 50%, 60%, 70%, 80%, and 90% points of testing) using all datasets. While the smaller values of ARE% and EP mean that the prediction accuracy of the methods is better, the smaller portion of RE values within a threshold indicates worse performance.

Table 3
NHPP based models used in this study.

Model name	Mean value function	Reference
Generalized Goel	$m(t) = a(1 - e^{-bt^c})$	Huang et al. (2003)
Goel-Okumoto	$m(t) = a(1 - e^{-bt})$	Huang et al. (2003)
Gompertz	$m(t) = ak^{b^t}$	Huang et al. (2003)
Inflectioin S-Shape	$m(t) = \frac{a(1-e^{-bt})}{1+\beta e^{-bt}}$	Huang et al. (2003)
Modified Duane	$m(t) = a\left(1 - \left(\frac{b}{b+t}\right)^c\right)$	Huang et al. (2003)
Musa-Okumoto logarithmic Poisson	$m(t) = a\ln(1 + bt)$	Musa and Okumoto (1984)
YID model1	$m(t) = \frac{ab}{b+\alpha}(e^{\alpha t} - e^{bt})$	Yamada et al. (1992)
YID model2	$m(t) = a(1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha at$	Yamada et al. (1992)
Delayed S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$	Lo and Huang (2006)
P-N-Z model	$m(t) = \frac{a}{1+\beta e^{-bt}}((1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha at)$	Nordmann (1999)
P-Z model	$m(t) = \frac{1}{1+\beta e^{-bt}}((c + a)(1 - e^{-bt}) - \frac{a}{b-\alpha}(e^{-\alpha t} - e^{-bt}))$	Pham (2006)
P-Z Imperfect fault detection	$m(t) = a - ae^{-bt} \times (1 + (b + d)t + bdt^2)$	Pham (2006)
Logistic growth curve model	$m(t) = \frac{a}{1+(ce^{bt})}$	Huang et al. (2003)

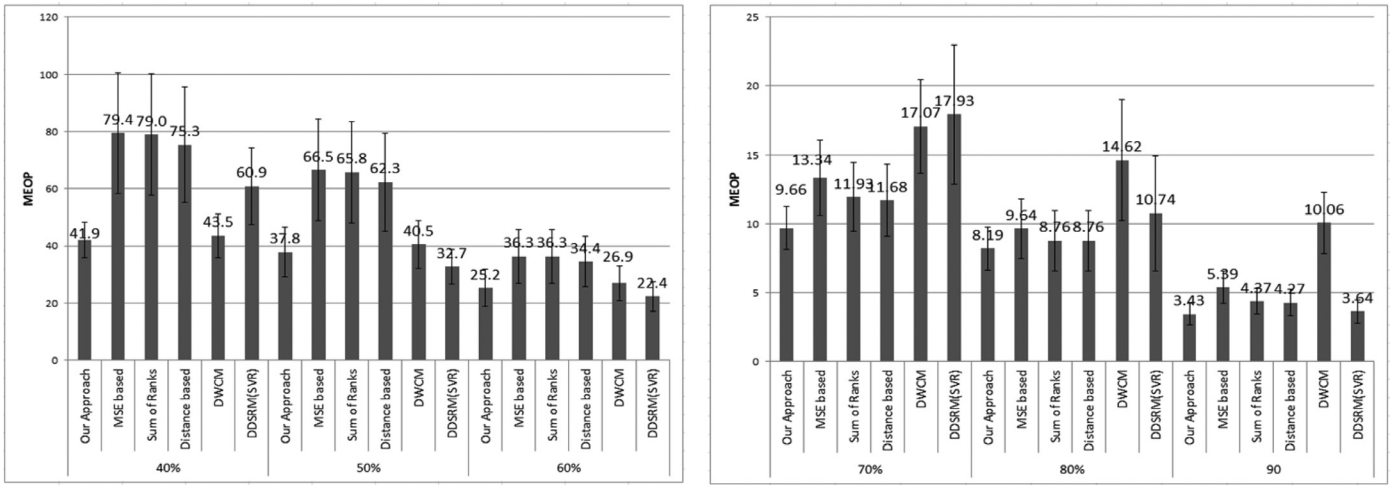

Fig. 5. Bar plots of MEOP values of six approaches at different testing periods.

6. Experimental results and analyses

6.1. Evaluation

This section describes the experimental results and analyses regarding the prediction performance of our decision tree based method and other methods. We compare the prediction performance of our method with that of some representative model selection, combination and data-driven modeling methods. Three different model selection methods are first compared. (i.e., (1) Naïve (MSE based) method, (2) sum of criteria ranks (Lyu and Nikora, 1992), (3) distance based method (Sharma et al., 2010)). Our approach is also compared with the dynamic weighted combinational model (DWCM) which is one of the representative combination models (Su and Huang, 2007). For the comparison with DDSRMs, we use the DDSRM that is a time series model based on support vector machine for regression (SVR) because some recent studies show that the SVR model outperformed other techniques in the software reliability prediction (Kumar and Singh, 2012; Moura et al., 2011). For the long-term prediction of DDSRM (SVR), we adopt the recursive strategy.

Through the cross validation technique, we evaluated the prediction performance of approaches on all datasets used in this study. We use three evaluation criteria mentioned in Section 5. We first compared MEOP and EP at the various testing points (i.e., at the 40%, 50–90% of testing), which enables to check the long-term and short-term prediction accuracies of approaches. Figs. 5 and 6 show the results of MEOP and EP criteria respectively using a bar plot which displays the average performance values of approaches with the corresponding standard deviation. Every bar in each plot compares the performance of the corresponding methods at different testing periods.

We first observed that the MEOP values of most approaches were decreased as testing proceeds. It is reasonable because the shorter prediction horizon and more data for model fitting mean the lower uncertainty. The MEOP tests on the thirty software failure datasets in Fig. 5 indicate that our approach gives relatively accurate and robust prediction than the others in the most proportion of testing periods (i.e., in long-term or short-term). On the other hand, the existing model selection methods that use multi-criteria (i.e., sum of criteria ranks or distance based method) do not show much improvement in prediction accuracy, compared to the naïve method. We also observed that the gaps between our approach and model selection methods are larger at the early stage of testing, which means that applying one criteria or multi-criteria with same weights is inadequate to make long-term prediction. The results from end-point prediction comparison were similar with those from the MEOP comparison. As shown in Fig. 6, the proposed approach generally gives a smaller error variance than any of the other methods, which shows that our approach can improve the capability of predicting the number of cumulative failures at the end of the test period.

We also compare the approaches on the portion of RE values within 5%. RE is computed at every partition point from 40% of testing. As shown in Table 4, overall result values are rather low due to the conservative setting of a threshold. Although our approach is competitive with DDSRM(SVR), our approach gives RE values less than or

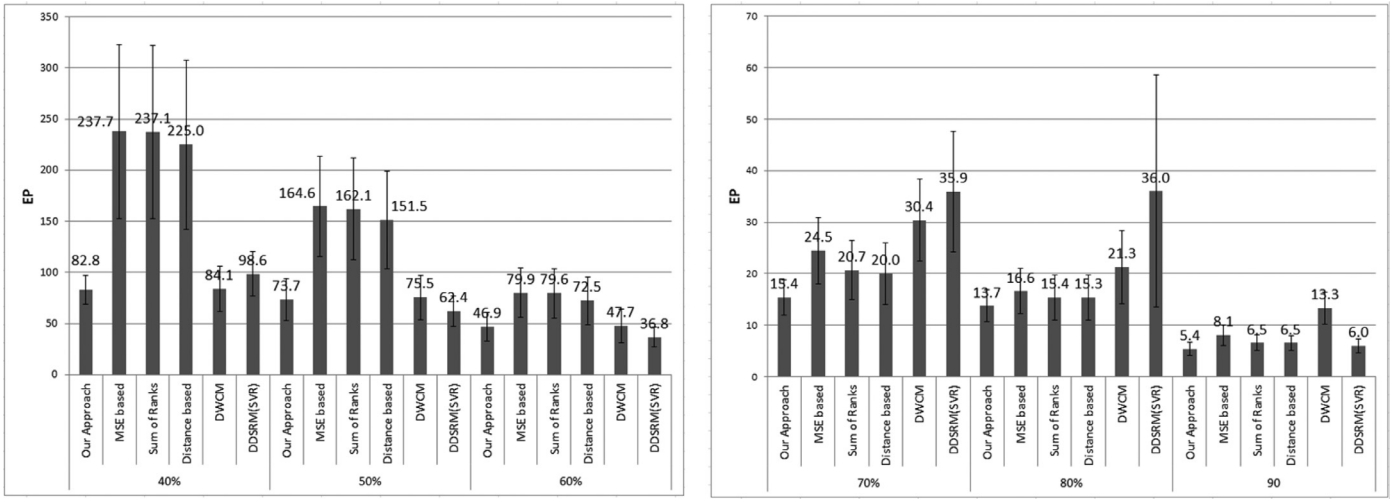

Fig. 6. Bar plots of EP values of six approaches at different testing periods.

Table 4
Prediction results (RE ≤ 5%) of approaches at different testing periods.

Partition %	Our approach	MSE based	Rank based	Distance based	DWCM	DDSRM (SVR)
40	12.6%	9.6%	9.7%	10.6%	8.5%	13.1%
50	22.3%	21.2%	20.3%	16.8%	8.6%	18.2%
60	26.2%	23%	24.7%	21.9%	19.6%	25.8%
70	43.1%	34.3%	37.9%	42.2%	39.3%	45%
80	55.7%	48.9	51.5%	45.1%	34.6%	68.3%
90	81.3%	56.4	61.6%	60.6%	45.5%	79.8%

similar to the values by other methods in most cases. This implies that the proposed method provides stable software reliability prediction in terms of the portion of RE values within 5%. These promising results indicate that the proposed decision tree based model selection and combination method can provide relatively robust and accurate prediction to practitioners.

6.2. Automated tool

We developed a tool which assists the proposed reliability prediction framework. The tool is developed based on Matlab/WEKA (Hall et al., 2009), and it can be downloaded at http://spiral.kaist.ac.kr/wp/research/reliability-tools/. Empirical failure datasets can be saved in EXCEL sheets, and the tool reads all failure datasets of a selected EXCEL file. This EXCEL file can be maintained as the repository of organization for software failure information.

The tool supports the software reliability prediction based on multi-criteria using empirical datasets. Although this is not a GUI based tool, it includes all functions to predict the future software failures.

6.3. Threats to validity

This section discusses the limitations of our work. They are usually related to generalizing our approach to industrial practice. Although thirty projects were applied in this study, their characteristics might be biased, and it accidentally causes better (or worse) results with some approaches. To eliminate such a bias, we collected and examined as much project data as possible that have been widely used in previous studies. The datasets used in this study can also be extended. The decision tree models trained with more datasets usually generate more accurate results. Therefore, we can expect the better prediction accuracy with more datasets due to the characteristics of our approach which uses historical data. It is recommended to accumulate more datasets, which may contribute to building more reliable decision trees and making more accurate prediction.

Other factors are criteria and models applied in the experiment. We selected the ten comparison criteria and thirteen models that are widely adopted and validated by previous studies regarding software reliability. Although there may be other important criteria and models, they can also be incorporated into our software reliability prediction framework and contribute much more to improving the accuracy of reliability prediction.

7. Conclusion

In this paper, we have proposed the decision tree based software reliability prediction framework. We trained the patterns of multi-criteria on observed data to characterize the existing SRGMs. The trained decision tree model generally included only a subset of the criteria that are deemed truly informative for prediction. This approach automatically assigns a weight to each model and infers the model which is likely to make the most trustworthy predictions depending on empirical evidences, rather than the subjective decision of practitioners. For the further improvement of reliability prediction, the framework combines the identified models with consideration of the assigned weights and the tendency of over- or under-prediction. We also developed the tool that assists the framework, which can automate the computation of criteria and the construction of decision trees using historical datasets. In the experiments using thirty datasets, the proposed framework showed fairly good prediction performance, which indicates relatively more accurate and robust software reliability prediction even at the long term as well as short term.

The trained decision trees can be used in any other reliability prediction works, and we can expect better prediction performance with more datasets. In future, we will investigate the applicability of the proposed framework to industrial projects and further explore if the framework improves the software reliability prediction in practice.

Acknowledgments

This work was partly supported by the ICT R&D program of MSIP/IITP [10044457, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices] and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2013R1A1A2006985).

References

ABDEL-GHALY, A.A., CHAN, P., Littlewood, B., 1986. Evaluation of competing software reliability predictions. Softw. Eng. IEEE Trans. SE-12, 950–967.
Amin, A., Grunske, L., Colman, A., 2013. An approach to software reliability prediction based on time series modeling. J. Syst. Softw. 86, 1923–1932.
Andersson, C., 2006. A replicated empirical study of a selection method for software reliability growth models. Empir. Softw. Eng. 12, 161–182.
Cai, K.-Y., Cai, L., Wang, W.-D., Yu, Z.-Y., Zhang, D., 2001. On the neural network approach in software reliability modeling. J. Syst. Softw. 58, 47–62.
Chiu, K.-C., Huang, Y.-S., Lee, T.-Z., 2008. A study of software reliability growth from the perspective of learning effects. Reliab. Eng. Syst. Saf. 93, 1410–1421.
Dharmasena, L.S., Zeephongsekul, P., Jayasinghe, C.L., 2011. Software reliability growth models based on local polynomial modeling with kernel smoothing. in: 2011 IEEE 22nd International Symposium on Software Reliability Engineering. IEEE, pp. 220–229.
Goel, A., 1985. Software reliability models: assumptions, limitations, and applicability. IEEE Trans. Softw. Eng. SE-11, 1411–1423.
Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA data mining software: an update. SIGKDD Explor. Newsl. 11 (1), 10–18.
Holmes, G., Pfahringer, B., Kirkby, R., 2002. Multiclass alternating decision trees. Mach. Learn. ECML 2430, 161–172.
Hu, Q., Xie, M., Ng, S., Levitin, G., 2007. Robust recurrent neural network modeling for software fault detection and correction prediction. Reliab. Eng. Syst. Saf. 92, 332–340.
Huang, C., Lyu, M., Kuo, S., 2003. A unified scheme of some nonhomogeneous poisson process models for software reliability estimation. IEEE Trans. Softw. Eng. 29, 261–269.
Huang, C.-Y., Kuo, S.-Y., Lyu, M.R., 2007. An assessment of testing-effort dependent software reliability growth models. IEEE Trans. Reliab. 56, 198–211.
Jeske, D., Zhang, X., Pham, L., 2005. Adjusting software failure rates that are estimated from test data. Reliab. IEEE Trans. 54, 107–114.
Jones, W., 1991. Reliability models for very large software systems in industry, in: Proceedings of the Second International Symposium on Software Reliability Engineering. pp. 35–42.
Karunanithi, N., 1992. Prediction of software reliability using connectionist models. IEEE Trans. Softw. Eng. 18, 563–574.
Karunanithi, N., Whitley, D., Malaiya, Y.K., 1992. Using neural networks in reliability prediction. IEEE Softw. 9, 53–59.
Kharchenko, V., Tarasyuk, O., Dubnitsky, V., 2002. The method of software reliability growth models choice using assumptions matrix, in: Proceedings of the 26th Annual International Computer Software and Applications Conference. pp. 541–546.
Khoshgoftaar, T., Woodcock, T., 1991. Software reliability model selection: a case study, in: Proceedings of the Second International Symposium on Software Reliability Engineering. pp. 183–191.
Konishi, S., Kitagawa, G., 2008. Various model evaluation criteria, in: Information Criteria and Statistical Modeling. Springer, New York, pp. 239–254.
Kumar, P., Singh, Y., 2012. An empirical study of software reliability prediction using machine learning techniques. Int. J. Syst. Assur. Eng. Manag. 3, 194–208.
Li, N., Malaiya, Y., 1993. Enhancing accuracy of software reliability prediction. Softw. Reliab. Eng. 1993.
Li, X., Li, Y.F., Xie, M., Ng, S.H., 2011. Reliability analysis and optimal version-updating for open source software. Inf. Softw. Technol. 53, 929–936.
Lo, J.-H., Huang, C.-Y., 2006. An integration of fault detection and correction processes in software reliability analysis. J. Syst. Softw. 79, 1312–1323.
Lyu, M., 1996. Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill, New York.
Lyu, M., Nikora, A., 1992. CASRE: a computer-aided software reliability estimation tool, in: Proceedings of the Third International Workshop on Computer-Aided Software Engineering. pp. 264–275.
Makridakis, S., Andersen, A., 1982. The accuracy of extrapolation (time series) methods: results of a forecasting competition. J. Forecast. 1, 111–153.
Moura, M.D.C., Zio, E., Lins, I.D., Droguett, E., 2011. Failure and reliability prediction by support vector machines regression of time series data. Reliab. Eng. Syst. Saf. 96, 1527–1534.
Musa, J., Iannino, A., Okumoto, K., 1987. Software Reliability: Measurement, Prediction, Application. McGraw-Hill, New York.
Musa, J., Okumoto, K., 1984. A logarithmic Poisson execution time model for software reliability measurement. ICSE Proceedings 7th Int. Conf. Softw. Eng., 230–238.
Nordmann, L., 1999. A general imperfect-software-debugging model with S-shaped fault-detection rate. IEEE Trans. Reliab. 48, 169–175.
Pai, P.-F., Hong, W.-C., 2005. Support vector machines with simulated annealing algorithms in electricity load forecasting. Energy Convers. Manag. 46, 2669–2688.
Pai, P.-F., Hong, W.-C., 2006. Software reliability forecasting by support vector machines with simulated annealing algorithms. J. Syst. Softw. 79, 747–755.
Pham, H., 2006. System Software Reliability, Springer Series in Reliability Engineering. Springer, London.
Pillai, K., Nair, V.S., 1997. A model for software development effort and cost estimation. Softw. Eng. IEEE Trans. 23, 485–497.
Raj Kiran, N., Ravi, V., 2008. Software reliability prediction by soft computing techniques. J. Syst. Softw. 81, 576–583.
Sharma, K., Garg, R., Nagpal, C.K., Garg, R.K., 2010. Selection of optimal software reliability growth models using a distance based approach. IEEE Trans. Reliab. 59, 266–276.
Singh, Y., Kaur, A., Malhotra, R., 2010. Empirical validation of object-oriented metrics for predicting fault proneness models. Softw. Qual. J. 18, 3–35.
Stringfellow, C., Andrews, A., 2002. An empirical method for selecting software reliability growth models. Empir. Softw. Eng. 7, 319–343.
Su, Y.-S., Huang, C.-Y., 2007. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. J. Syst. Softw. 80, 606–615.
Tapio Elomaa, M.K., 2001. An analysis of reduced error pruning. J. Artif. Intell. Res. 15, 163–187.
Tian, L., Noore, A., 2005a. Evolutionary neural network modeling for software cumulative failure time prediction. Reliab. Eng. Syst. Saf. 87, 45–51.
Tian, L., Noore, A., 2005b. Dynamic software reliability prediction: an approach based on support vector machines. Int. J. Reliab. Qual. Saf. Eng. 12, 309–321.
Tohma, Y., Jacoby, R., 1989. Hyper-geometric distribution model to estimate the number of residual software faults, in: Proceedings of the 13th Annual International Computer Software and Applications Conference. pp. 610–617.
Venables, W., Ripley, B., 1994. Modern Applied Statistics with S-PLUS. Springer Verlag, USA.
Wood, A., 1996. Predicting software reliability. Computer (Long. Beach. Calif) 29, 69–77.
Wood, A., 1997. Software reliability growth models: assumptions vs. reality, in: Proceedings of the Eighth International Symposium On Software Reliability Engineering. pp. 136–141.
Xiao, X., Dohi, T., 2013. Wavelet shrinkage estimation for non-homogeneous Poisson process based software reliability models. IEEE Trans. Reliab. 62, 211–225.
Yamada, S., Tokuno, K., Osaki, S., 1992. Imperfect debugging models with fault introduction rate for software reliability assessment. Int. J. Syst. Sci. 23, 2241–2252.
Yang, B., Li, X., Xie, M., Tan, F., 2010. A generic data-driven software reliability model with model mining technique. Reliab. Eng. Syst. Saf. 95, 671–678.
Zhang, X., Pham, H., 2006. Software field failure rate prediction before software deployment. J. Syst. Softw. 79, 291–300.
Zhang, X., Teng, X., Pham, H., 2003. Considering fault removal efficiency in software reliability assessment. Syst. Man Cybern. Part A Syst. Humans, IEEE Trans. 33, 114–120.
Zhao, M., Xie, M., 1992. On the log-power NHPP software reliability model, in: Proceedings of the Third International Symposium on Software Reliability Engineering. pp. 14–22.
Zhou, Y., Leung, H., 2007. Predicting object-oriented software maintainability using multivariate adaptive regression splines. J. Syst. Softw. 80, 1349–1361.

Jinhee Park is a Ph.D. candidate in the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST). He received his Master's degrees in Computer Science from KAIST. His research interests include software reliability estimation, software measurement and analysis, mining software repositories, and software process improvement.

Jongmoon Baik received his M.S. degree and Ph.D. degree in computer science from University of Southern California in 1996 and 2000 respectively. He received his B.S. degree in computer science and statistics from Chosun University in 1993. He worked as a principal research scientist at Software and Systems Engineering Research Laboratory, Motorola Labs, where he was responsible for leading many software quality improvement initiatives. Currently, he is an associate professor in the Computer Science Department at Korea Advanced Institute of Science and Technology (KAIST). His research activity and interest are focused on software six sigma, software reliability & safety, and software process improvement.