Łukasz Radliński[†‡] Norman Fenton[‡] Martin Neil[‡]
[†]Instytut Informatyki w Zarządzaniu, Wydział Nauk Ekonomicznych i Zarządzania, Uniwersytet Szczeciński, ul. Mickiewicza 64, 71-101 Szczecin, Poland
[‡]Department of Computer Science, Faculty of Informatics and Mathematical Sciences, Queen Mary, University of London, Mile End Road, London E1 4NS, United Kingdom

A Learning Bayesian Net for Predicting Number of Software Defects Found in a Sequence of Testing

Abstract

We present the model for predicting the number of defects found in each testing iteration. The model assumes that the software is developed either according to the classical waterfall lifecycle or its variation, where testing takes place in a series of iterations after the coding of the whole software is finished. The input data to the initial model are the numbers of defects found in some first iterations of testing. Our model can also use as the input the estimated total number of residual defects given by other models. We partially validate the model using the publicly available datasets of software projects developed in cooperation with NASA: JM1, KC1, PC1, PC3, PC4. However, these datasets did not contain any information on testing effort and our results highlight the importance of recording such data for factors influencing testing effectiveness. We show how the model can be extended by adding other input variables such as effort, process and people quality and bias. Finally, we show the results of validating this extended model using a semi-randomly generated dataset.

Keywords: defect prediction, learning Bayesian net, testing sequence

Background and motivation

Software quality prediction has been a well researched topic in software engineering for many years. This research has generally focused on a) predicting the total number of residual defects remaining in software after testing or release [1, 2, 3, 4]; b) identifying fault prone components (module, class etc.) [5, 6, 7]; c) reliability treated as predicting time between failures or similar measures [8, 9]; and d) predicting the number of defects to be found in specific time intervals in the future. Increasingly we have found that practitioners are most concerned with this latter problem. Yet none of the past achievements in this area can be regarded as totally satisfactory. In this paper we describe a model (a Bayesian net) that we developed to predict the number of defects that are likely to be found in future testing iterations, given data about defects on previous iterations. This problem has been analyzed in numerous studies using different modeling techniques [10, 11, 12, 13, 14]. We also discuss the initial model validation results.

Generally, we can develop 3 types of models for predicting number of defects in a specific testing iteration:

1. Models following a specific trend line.

Before developing a model a trend type has to be determined. Then an expression that is a function of time has to be developed. At the end the parameters for this expression describing the trend line have to be estimated using some of the data from the available dataset. The prediction is made by entering the iteration number as an input to the expression (possibly with other variables describing the testing process).

2. Learning models following a specific trend line.

As in the previous type both a trend type and an expression describing the trend have to be determined. But the expression parameters do not need to be estimated outside the model. Rather the values of early observations of defects found are entered in the model. Then the model learns its parameters – in BNs this is done using back-propagation. As in the previous type, the prediction is made by entering the iteration number as an input to the model (also possibly with other variables describing testing process).

3. Learning models without the specific trend line assumption.

In this type of model there is no assumption about the type of trend line followed by the data. It can be modeled as a network – set of variables influencing each other. The unknown values of variables are learnt after entering observations for number of defects found in some early testing iterations. The prediction is made after generating more instances of single iterations and linking them in the same way as they were defined for learning.

In this study we analyze the possibility of developing a model of the latter type – without the specific trend line assumption for defects found. We want to develop a model usable in various testing approaches without assumptions about the trend lines. Furthermore, we want to expand the model by other actions and consequences related to testing such as: fixing defects and introducing new defects as a result of imperfect fixes.

The datasets

In our study we have used some of the publicly available datasets of projects developed in cooperation with NASA [15]: JM1, KC1, PC1, PC3, PC4. These were the only NASA datasets that met the following criteria:

- defects have assigned a date of finding – datasets containing large number defects which do not have assigned date of finding were excluded from the analysis as we cannot calculate the aggregated number of defects found in a testing iteration,
- the values follow some kind of a trend – there may be some deviations from the trend line but generally some kind of a trend is followed,
- the testing phase lasted for some time (at least 30 iterations) – so that our model can learn using some early observed values and predict the future,

We decided to perform analysis for testing iteration of 1 month. Although it might be more practical to have shorter iteration duration, e.g. 1 week, we found that the datasets were too noisy. The model could not learn any reasonable trend from such noisy data. Even if the model somehow could be trained using such data its predictions would be very different from actual values observed in iterations after learning the model. It would predict only a general trend learnt but, without any other variables explaining variation of observed defects found, not the actually observed values.

The monthly defect data from NASA generally follow a so called 'delayed S' trend line [10, 16] where the number of defects found per iteration increases at the beginning and after reaching its peak slowly decreases toward 0. However, there are a couple of values far away from the general trend line in each dataset. But the variation between subsequent iterations is much smaller in monthly aggregations than in the case of weekly data.

The initial model

We assume that the model can be used in the environment where the software is developed in a classical waterfall cycle. The model assumes that there is some unknown number of residual defects introduced before testing. It also assumes that no new defects will be introduced once the testing has started which means that no new functionality will be added to the code base. The testing process is not treated as a black-box where we cannot identify its parts. Rather, we treat a whole testing process as a set of testing iterations repeated one after another.

The model consists of the following parts (Figure 1):

1. **Priors**.

This subnet contains initial (prior) probability distributions for variables that are to be learnt later: testing effectiveness and residual defects.

2. **Learning**.

This subnet contains sets of variables linked together as a single BN. Its purpose is to learn the unknown residual defects after each iteration and testing effectiveness in each iteration.

3. **Prediction**.

The first *k* iterations (e.g. 3 as illustrated on Figure 1) are used for learning the model – we have to enter observations for defects found in the first *k* testing iterations. Starting from iteration *k+1* the model predicts the number of defects found which are likely to occur in each future testing iteration. It also predicts the testing effectiveness in each testing iteration and number of residual defects remaining after each future testing iteration. For reasons of computational efficiency, this part of the model is a Dynamic Bayesian Net (DBN) with testing iterations linked sequentially. Here the model does not learn anything from the data, but rather uses the previously learnt testing effectiveness and number of residual defects to predict the number of defects found in the future.

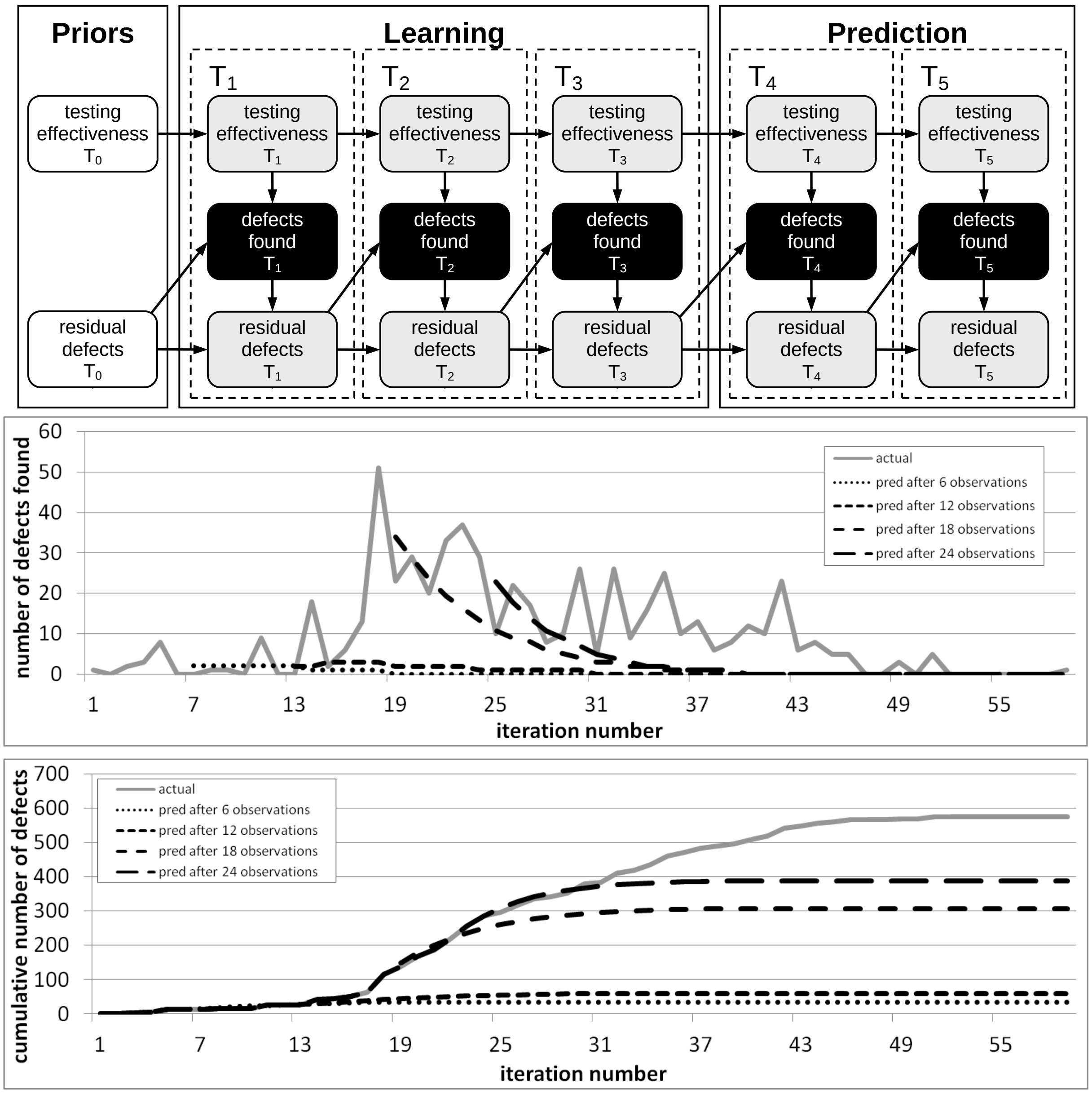

Figure 1 The detailed structure of the testing iterations (top) and initial prediction results with PC1 dataset (middle and bottom)

We entered the following expression in the model variables:

- *testing effectiveness T_0 = Uniform(0, 1),*
- *residual defects T_0 = Normal(1000, 1000000).*
- *testing effectiveness T_i = TNormal(testing effectiveness T_{i-1}, 0.001, 0, 1).*
- *residual defects T_i = Max(0, residual defects T_{i-1} – defects found T_i).*
- *defects found T_i = Binomial(residual defects T_{i-1}, testing effectiveness T_i).*

We have developed and executed our model using the API for the AgenaRisk toolset [17].

Lessons learned from model creation and validation

1. Influence of the prior distribution in 'residual defects'

There are two important parameters in the expression in this variable: its mean and the variance. Both of them may have a significant impact on the predictions given by the model. Ideally we would like to have a point value distribution in this node reflecting the true number of residual defects with no variance (certain value). But we know it is impossible to estimate the number of residual defects with 100% accuracy using any method. Still, it is important to set the distribution of this node with a mean as close to the real number of residual defects and the variance as low as possible. Setting a variance to a very low value means that the model has more 'trust' in the entered distribution in this variable than the observations entered for defects found in some first testing iterations. This is especially important when the mean value in this variable has been set far from the real number of the residual defects. In such cases the model under- or overestimates the predicted number of defects found (in the iterations following those for which we entered observations in defects found) and the number of defects remaining after a specific testing iteration.

2. Influence of the variance in expression in 'testing effectiveness'

The prediction for number of defects found in future testing iterations depends on the value of the variance entered in the variable 'testing effectiveness'. If the variance entered is low (e.g. 0.0001) the model predicts that the defects will be found in the longer period of time. If the variance is high (e.g. 0.05) then in many scenarios the model predicts that few or no defects will be found in the testing iterations following the last iteration with entered observation in the number of defects found. The reason for such contrasting predictions is that in the second case (higher variance) the model believes that the majority of the defects have already been found during past testing iterations. Higher than usual values in the number of defects found in specific iteration are explained in the model by the higher testing effectiveness in a particular iteration. Such explanations can be given by the model because with higher variance in the equation the model can 'understand' higher variation in the real testing effectiveness which might have been caused by allocating more effort and/or more testers taking part in the process of finding defects. In the scenarios that we tested we found that the most reasonable predictions can be given when the variance is set to value around 0.001.

3. Influence of the prior distribution in 'testing effectiveness'

The variable 'testing effectiveness' reflects the probability of finding a defect in a specific testing iteration. Because of this assumption it must be within the range from 0 to 1. But without any other process factors in the model we cannot make any further assumptions about its value. For example, we cannot say if the value of 0.05, meaning that we should expect around 5% of the defects to be found during specific iteration, is high or low. It depends on the granularity of the data we use in our model. If the testing iteration is brief, e.g. one day, then such a value appears to be attractively high. On the other hand, if the testing iteration is long, (e.g. three months), than it does not seem so attractive anymore. Clearly, the value also depends on the software itself, especially the size which influences the number of defects the most. The more residual defects are in the software the less likely it is that the testers will find the majority of them in a single iteration. It all means that we cannot make any assumption about the prior probability distribution in the 'testing effectiveness'. That is why we entered the Uniform distribution over the range from 0 to 1 in which any of the values within this range is equally likely to happen. After entering the observations to defects found in some first testing iterations the distributions in this variable in all iterations are revised – they are not that flat anymore.

4. Influence of the expression in 'defects found'

In our model the number of defects found is expressed as a Binomial distribution with number of trials equal to remaining residual defects before the specific testing iteration and with probability of success equal to the testing effectiveness in the current iteration. Such a distribution has been used in earlier models [2, 18].

5. Prediction accuracy

The basic version of the model does not give accurate predictions for number of defects found in future iterations. This can be explained by the lack of variables in the model which would be able to explain the variation in the number of defects found between iterations, i.e. why the testing effectiveness varies between subsequent iterations so much. However, given the observations entered to the model, we actually get reasonable predictions. The model is highly inaccurate especially for entering observations for small number of iterations (<10). But if the values entered are very low (as they usually are in the analyzed datasets) why should the model predict the increase of number of defects found in future iterations without any information about the number of residual defects and, more importantly, the factors affecting the testing effectiveness? To have improved and useful predictions we need the extended model with additional variables.

Extending the model

To improve the model's accuracy and realism we added new variables to the model. Two factors seem to be most influential on the testing effectiveness:

- effort spent on testing,
- testing process and people quality.

These new variables are not expressed on an absolute scale. Rather they reflect the ratio of the value in the given iteration to the value in the previous iteration. We assume that users can provide observations for these variables in the first iterations used to learn the model. For example, if the effort in the current iteration increased by 20% compared to the previous one the users should enter a value '1.2'.

Although effort and process and people quality seem to be most influential on the testing effectiveness they do not explain the whole variation of the testing effectiveness. In certain cases it may happen that although both effort and process and people quality increased we observe the lower testing effectiveness. Therefore we added yet another variable 'bias' which reflects the aggregation of all negative factors occurring in the specific testing iteration. This includes, for example, the need to prepare additional test cases (which causes the situation whereby some effort is not effectively used on purely finding defects) or testing a component that was not tested much before (and testers need to learn this component to know where and how it should be tested).

We added an intermediate node 'testing effectiveness change T_i' which aggregates the testing factors. It reflects the extent at which the testing effectiveness changed in the given iteration compared to the previous one. This node is a parent node for 'testing effectiveness T_i' together with the 'testing effectiveness T_{i-1}' and 'testing effectiveness limit'. Introducing the latter ensures that the change of testing factors to some degree does not cause the same degree of change in testing effectiveness but lower. The structure of the single iteration in the extended model is illustrated on Figure 2 (top part). The expressions in the new or updated nodes are the following:

- *testing effort T_i = Normal(1, 0.1),*

- *process and people quality T_i = Normal(1, 0.1),*
- *testing bias T_i = Normal(1, 0.1),*
- *testing effectiveness change T_i =*
  *Normal(process and people quality T_i * ((effort T_i – 1) * 0.8 + 1) / bias T_i, 0.01),*
- *testing effectiveness limit T_i = testing effectiveness limit T_{i-1},*
- *testing effectiveness T_i = testing effectiveness T_{i-1} **
  *((testing effectiveness change T_i – 1) * testing effectiveness limit T_i) + 1).*

The expression for 'testing effectiveness change T_i' ensures that testing effort has lower impact on the testing effectiveness than the process and people quality.

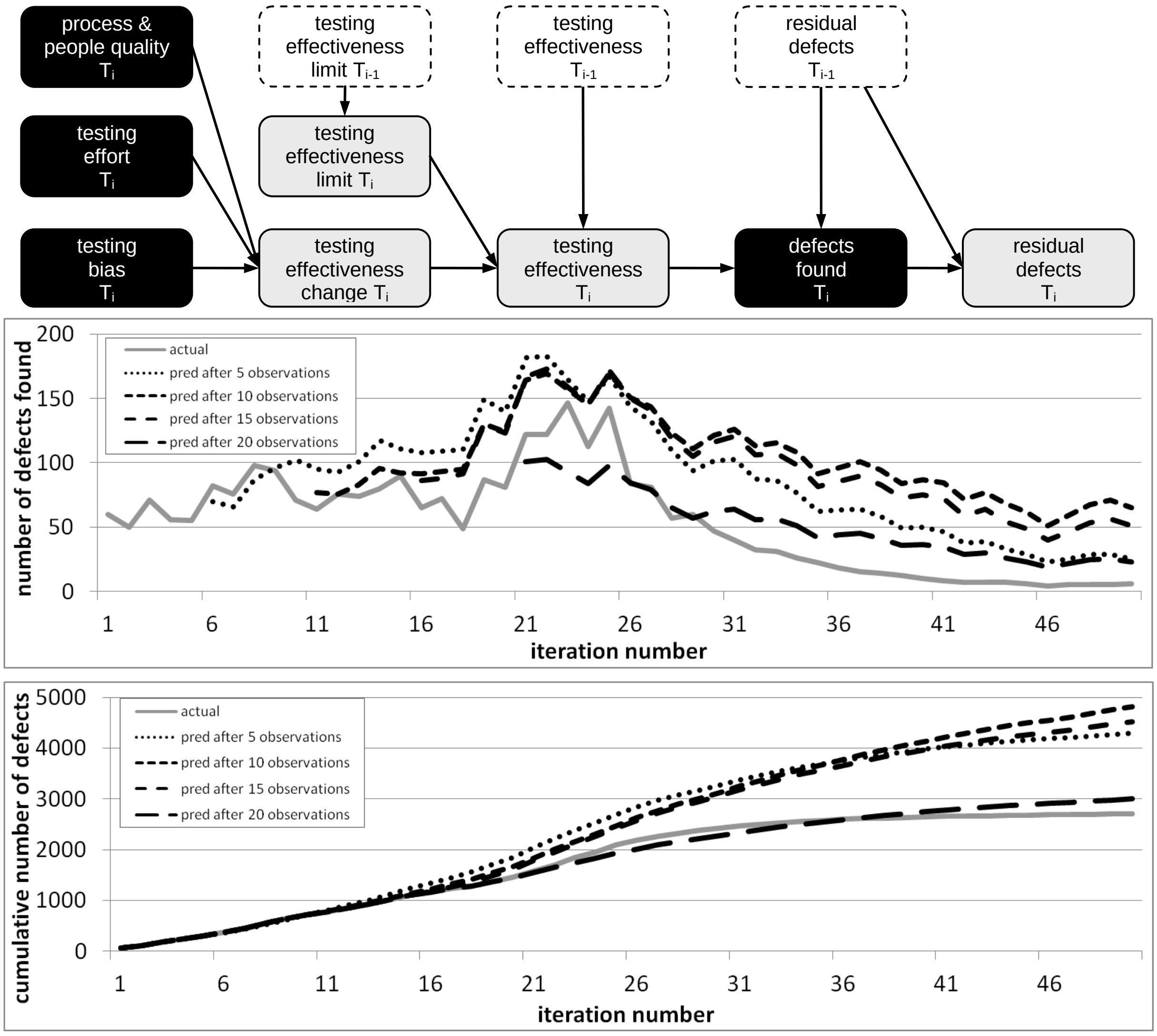

Figure 2 The structure of the extended model (top) with example prediction results (middle and bottom)

The observations in the extended model should be assigned for all known variables (effort, process and people quality, bias, defects found) in the first iterations used for learning the model. For prediction only the observations for effort, process and people quality and possibly for bias. These observations are treated as anticipated values – planned to be achieved. Based on these observations the model predicts the number of defects found in future testing iterations.

The NASA datasets used earlier do not contain any values reflecting testing effectiveness like testing effort or process and people quality. Therefore we do not use these datasets to validate an extended version of the model. Instead we generated a sample dataset sorely for the purpose to validate the extended version of the model. When generating this dataset we set an initial number of residual defects and generated random values for changes in effort, process and people quality and bias. Then, using these values and equations as in the model we generated the values for the number of defects found in each iteration. Then we manually changed some of the values to ensure that the shape of the trend line follows the one typically occurring in real projects (initial increase of the number of defects found, followed by a decrease towards 0).

We used this dataset by entering the values describing each iteration to the model. We did not enter the precise value of number of residual defects as in practice this would not be possible. Also, the values for effort, process and people quality and bias we entered with lower precision – in the dataset they were up to the second decimal place, in the model we entered these values rounded to the first decimal place. This is to simulate the accuracy of estimating these factors in real projects which will never be exactly 100% accurate.

The aim of this validation is to find out how fast the model learns the testing effectiveness and real number of residual defects which were in at the beginning of the testing process. Our results (Figure 2) show that the prediction accuracy for the first couple of iterations following the iterations used for learning is generally acceptable in every case. However, predicting the number of defects found far in the future requires more iterations used to learn the model.

Conclusions

We developed a Bayesian net model for predicting number of defects likely to be found in future testing iterations. We discussed various issues that arose during model development and validation which may be useful to other researchers building similar models and to practitioners needing to adjust the model to their needs.

We initially tested this model using some NASA datasets. This initial validation proved that the model cannot achieve high prediction accuracy without either incorporating the shape of the trend line of the number of defects found into the model or without additional factors influencing the testing effectiveness. We chose the latter option and included additional factors to the model: effort, process and people quality and bias.

The validation performed on a semi-randomly generated dataset proves the need for providing additional factors influencing testing effectiveness to such type of model. In the future we plan to extend the model to enable prediction for number of defects fixed in specific iteration. This will also require the changes in the model to capture the possibility of inserting new defects due to imperfect fixing.

Bibliography

1. Chulani S., Boehm B.: *Modelling Software Defect Introduction and Removal: COQUALMO (COnstructive QUAlity MOdel)*, Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering, 1999.
2. Fenton N.E., Neil M., Hearty P., Marsh W., Marquez D., Krause P., Mishra R.: *Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets*, Information & Software Technology, Vol. 49, 2007, pp. 32-43.
3. Gaffney J.R.: *Estimating the Number of Faults in Code*, IEEE Trans. Software Eng. Vol. 10, No. 4, 1984, pp. 141-152.
4. Lipow M.: *Number of Faults per Line of Code*, IEEE Trans. Software Eng., Vol. 8, No. 4, 1982, pp. 437-439.
5. Bell R.M., Weyuker E.J., Ostrand T.J.: *Predicting the Location and Number of Faults in Large Software Systems*, IEEE Trans. Software Eng., Vol. 34, No. 4, Apr. 2005, pp. 340-355.
6. Briand L.C., Melo W.L., Wüst J.: *Assessing the Applicability of Fault-Proneness Models across Object-Oriented Software Projects*, IEEE Trans. Software Eng., Vol. 28, No. 7, July 2002, pp. 706-720.
7. Pai G.J., Dugan J.B.: *Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods*, IEEE Trans. Software Eng., Vol. 33, No. 10, Oct. 2007, pp. 675-686.
8. Lyu M.R.: *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.
9. Musa J.D., Iannino A., Okumoto K.: *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1989.
10. Almering V., van Genuchten M., Cloudt G., Sonnemans P.J.M.: *Using Software Reliability Growth in Practice*, IEEE Software, Vol. 24, No. 6, 2007, pp. 82-88.

11. Cangussu J.W., DeCarlo R.A., Mathur A.P.: *A Formal Model of the Software Test Process*, IEEE Trans. Software Eng., Vol. 28, No. 8, Aug. 2002, pp. 782-796.
12. Kapur P.K., Goswami D.N., Bardhan Amit, Singh Ompal: *Flexible Software Reliability Growth Models with testing effort dependent learning process*, Applied Mathematical Modelling, in press, 2007, doi:10.1016/j.apm.2007.04.002.
13. Levendel Y.: *Reliability Analysis of Large Software Systems: Defect Data Modelling*, IEEE Trans. Software Eng., Vol. 16, No. 2, Feb. 1990, pp. 141-152.
14. Zhao Jianmin, Chan A.H.C., Roberts C., Madelin K.B.: *Reliability Evaluation and optimization of imperfect inspections for a component with multi-defects*, Reliability Eng. and System Safety, Vol. 92, 2007, pp. 65-73.
15. Metrics Data Program, NASA IV&V facility, 2007, http://mdp.ivv.nasa.gov/.
16. Kan S.H.: *Metrics and Models in Software Quality Engineering. Second Edition*, Addison-Wesley Longman Publishing Co., Inc., Boston, 2002.
17. Agena: *AgenaRisk. Bayesian Network Software Tool*, www.agenarisk.com, 2008.
18. Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P.: *Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction*, Proc. Third Int. Workshop on Predictor Models in Software Eng., Int. Conf. on Software Eng., IEEE Computer Society, Washington, DC, 2007, p. 2.