CSCanada
Studies in Sociology of Science
Vol. 5, No. 2, 2014, pp. 89-95
DOI:10.3968/4845

ISSN 1923-0176 [Print]
ISSN 1923-0184 [Online]
www.cscanada.net
www.cscanada.org

The Software Reliability Increase Method

Svetlana A. Yaremchuk[a],*; Dmitry A. Maevsky[a]

[a]Institute of Electro Mechanics and Electrical Management, Odessa National Polytechnic University, Odessa, Ukraine.
*Corresponding author.

Received 5 February 2014; accepted 26 April 2014
Published online 26 May 2014

Abstract
Our investigation purpose is to create the software reliability increase method. The proposed method allows creators to calculate statistic, probabilistic and valuating reliability indices of software components which contain defects. The method's aim is to take into consideration the statistic components complexity by means of composite metrics. The use of received indices provides for components finding which contain much more defects for refactoring and the first testing process. It contributes to increase identified and corrected defects quantity and improve the software reliability on average about 8%.

Key words: Software reliability; Complexity Software components; Defects; Predictable reliability indexes; Refactoring; Components testing

Svetlana A. Yaremchuk, & Dmitry A. Maevsky (2014). The Software Reliability Increase Method. *Studies in Sociology of Science, 5*(2), 89-95. Available from: URL: http://www.cscanada.net/index.php/sss/article/view/4845
DOI: http://dx.doi.org/10.3968/4845

INTRODUCTION

The use of human beings software quantity grows up continuously. The American analytical company «Gartner» evaluations testify that the development and maintenance software expenditures are about 2, 8 trillion dollars in 2013 which is more than 4 % in the previous year. At the same time the mankind's dependence of software quality grows up.

According to the standard (ISO/IEC 25010, 2010) the reliability is one of the eight major characteristics of software quality. The priori evaluation of software reliability is determined during the development phase before its functioning as total amount of the software defects. The total amount of defects can be calculated on bases of models and methods which are offered in the works of Maevsky (Maevsky & Yaremchuk, 2012).

In addition to the reliability evaluation the real number of defects allows test-managers to plan the required testing resources (the period of time, number of testers, units of equipment and software packages) for testing implementation within the limits of the planned term and budget. However these steps do not assist to reveal much more defects amount and increase reliability.

The undetected defects are very expensive for people. Multibillion losses, accident and disasters are well-known and described by Neumann (Neumann, 1995) which were caused by defects existence in software critical function systems. That's way it's necessary to detect as more defects as possible. For this purpose it's necessary to know detailed software reliability indices and in particular what components contain defects, what the probability of defects existence is and what defects quantity is in each component.

These development characteristics use permits to carry out valid and well-timed refactoring and during the component testing to reveal as more defects as possible. All this provides for the software reliability increase. That is why the software reliability increase method creating is a relevant and actual task.

1. CURRENT PROBLEM'S STATUS

The standard (IEEE 610.12:1990) determines the component (module) as separate discrete identifiable structural software unit. The software components substantially differ in size, complexity and amount of defects.

The software components differ in complexity when the size is equal. At the same time the components of the identical complexity or size may contain the different amount of defects. Many experts find the software complexity as an important and objective factor determining the probability of the defects occurrence. At the same time the amount of defects dependence on the software complexity is an under-researched and problematic question.

The well-known methods prediction analysis of the defects amount in the software components permit to divide them into four classes (Figure 1).

The authors of the article (Ma, *at al.*, 2007) used the algorithms of machine learning for defects prediction in the Software components. But these algorithms make a low degree evaluation of accuracy. The authors of the article (Mahaweerawat, *at al.*, 2002) used the fuzzy logic algorithms. These methods do not guarantee the needed accuracy when this model is not so tough. The authors (Thwin, *at al.*, 2005) also used neural networks.

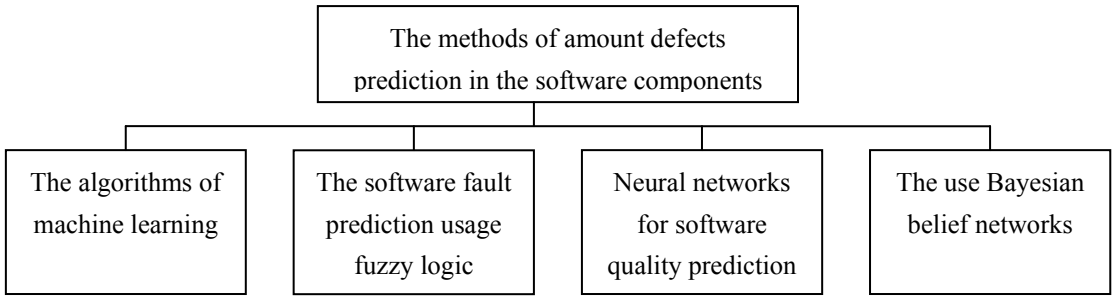

Figure 1
The Methods of Amount Defects Prediction in Software Components

Their main problem is that they have not explained why the given result had been obtained. That is why it is impossible to understand the reasons of defects occurrence and to take actions on the situation improvement. Besides, it exists the problem of so called «catastrophic forgetting» which explains the loss or distortion of the initially learned information in the process of the network retraining.

The Ukrainian experts O. Pomorova and T. Hovorushchenko (Pomorova, *at al.*, 2012) were confronted by difficulties to use in-built functions MatLab packet treatment metric's value without losing the significant information when researching the peculiarities of the software metric's ranging.

Another article authors (Fenton, *at al.*, 1999) investigated the methods which are based on the Bayesian belief networks. This method has a lot of disadvantages such as the necessity the experts recruitment, subjectivity of expert's evaluation, the full automatization complexity and high presenting work content.

Thus the analysis of well-known methods revealed the problems of low accuracy, and particularization of valuated software reliability indices, high complexity and work content, impossibility to understand the reasons of defects occurrence and also the necessity of experts recruitment.

These problems may be resolved by means of creating and using another method based on software complexity accounting. As distinct from well-known methods, the new one must:

provide the higher accuracy and particularization of valuated reliability indices

be simpler being used by the software company engineers without having a vast knowledge of higher mathematics

be based on accounting indices without using expensive experts' evaluations.

These presuppositions determine the purposes and tasks of present research.

2. PURPOSES AND TASKS OF THE RESEARCH

The purpose of the following article is to increase the software reliability by using detailed indices of components containing defects while refactoring and testing process.

It is necessary to work out a new method to obtain and apply needed indices.

To achieve the purpose it is necessary to solve the following **problems:**

a. To define the assumptions of the method;

b. To describe the method as a set of phases and steps;

c. To describe the example of the use of this method;

d. To make necessary conclusions for accuracy of the method estimation and increasing software reliability.

3. THE RESULTS OF THE RESEARCH

3.1 The Assumptions of the Method

The assumptions of the method are based on following hypotheses. In the software engineering the middle

branches or intra company reliability indices are very often used which had been obtained while developing preceding projects. However the investigations showed that the complexity of the software components and defects amount dependency in different projects of one developer are unique.

So the appreciation of indices of defects containing components based on the middle branches data or previous projects' ones may have significant digression from actual values. It is therefore appropriate to use the indices of the developing and valuating software for increasing accuracy appreciation.

The offered method is based on the following assumptions:

1) The defects amount in the Software components depends on the complexity of these components;

2) Other influencing factors do not change in the developing process of the software;

3) The components complexity may be quantitatively expressed by means of composite metric values of static program code complexity;

4) The space of simple events $\Omega = \{\omega_0, \omega_1, \omega_2, ...\omega_n\}$ for the probabilistic rates calculation of components containing defects are those facts which detect components without defects (ω_0), with only one defect (ω_1), with two of them (ω_2), …, with n defects (ω_n). In this case $\sum_{i=0}^{n} P_{\omega_i} = 1$;

5) The indices of the components containing defects are identical within a single software project. That is why the fined results of the components containing defects and defects of one already tested part of the project may be used for prediction indices accounting.

3.2 The Method Description

This method supposes the segmentation of all the components of the project A_{proj} for two subsets: basic A_{base} and predictive A_{pred} with the following qualities $A_{proj} = A_{base} \cup A_{pred}$, $A_{base} \cap A_{pred} = 0$.

It is necessary to determine and test the subsets of the basic components A_{base}. For each component it is necessary to estimate its complexity which is expressed by the composite metric value on basis of standard generally known metrics.

For the purpose of decreasing the method's work content it is proposed to divide the diapason of composite metric value into plenty of intervals $INT = \{\text{int}_k \mid k = \overline{1,m}\}$ and to group the components on intervals.

The revealed quantity of the components containing defects allow to get the reliability indices for all intervals of the complexity. Then the estimation of reliability indices for predictive subset A_{pred} is carried out on their basis.

The method's algorithm consists of the sequence of four stages and twenty steps.

Stage 1. Testing A_{base}, of the Software components complexity determination.

To determine A_{base}. For this purpose it is necessary to choose about ten components for each standard complexity metrics and to test A_{base}. To obtain the composite metric based on the standard metrics values and the number of defects in the components A_{base}.

To estimation the complexity of each component by means of the composite metric value.

To divide the diapason of complexity value into intervals.

Stage 2. The counting of initial data for basic subset A_{base}

a. To count the total components amount $NB_{\text{mod}}^{\text{int}_k}$.

b. To count the amount of the components containing defects $NB_{\text{mod\_}def}^{\text{int}_k}$ and also the components with *1,2,3,i*... defects $NB_{\text{mod\_}i\_def}^{\text{int}_k}$.

c. To count the total amount of defects $NB_{def}^{\text{int}_k}$. To count the total amount of the components $NP_{\text{mod}}^{\text{int}_k}$ for predictive subset A_{pred}.

Stage 3. The calculation of probabilistic, statistic and estimated reliability indices

The probability of existence of the component containing defects into complexity intervals

$$P_{\text{mod\_}def}^{\text{int}_k} = \frac{NB_{\text{mod\_}def}^{\text{int}_k}}{NB_{\text{mod}}^{\text{int}_k}} \quad (1)$$

The probability of existence in components 1,2,3,*i*... defects

$$P_{\text{mod\_}i\_def}^{\text{int}_k} = \frac{NB_{\text{mod\_}i\_def}^{\text{int}_k}}{NB_{\text{mod\_}def}^{\text{int}_k}} \quad (2)$$

The average number of defects in the components

$$\overline{N_{def}^{\text{int}_k}} = \frac{NB_{def}^{\text{int}_k}}{NB_{\text{mod}}^{\text{int}_k}} \quad (3)$$

The estimation of the component containing defects in the complexity intervals

$$N_{\text{mod\_}def}^{*\text{int}_k} = NP_{\text{mod}}^{\text{int}_k} \cdot P_{\text{mod\_}def}^{\text{int}_k} \quad (4)$$

and the estimation of the amount of the components containing defects in the predictive subset A_{pred}

$$N_{\text{mod\_}def}^{*} = \sum_{k=1}^{m} N_{\text{mod\_}def}^{*\text{int}_k} \quad (5)$$

The estimation of the amount of defects in the complexity intervals

$$N_{def}^{*\text{int}_k} = NP_{\text{mod}}^{\text{int}_k} \cdot \overline{N_{def}^{\text{int}_k}} \quad (6)$$

and the estimation of defects amount in the predictive subset A_{pred}

$$N_{def}^{*} = \sum_{k=1}^{m} N_{def}^{*\text{int}_k} \quad (7)$$

The code base defectiveness degree

$$CBDD = \frac{\sum_{k=1}^{m} NB_{\text{mod}\_def}^{\text{int}_k}}{\sum_{k=1}^{m} NB_{\text{mod}}^{\text{int}_k}} \cdot 100\% \quad (8)$$

g. The criterion of the necessary reliability achievement when the recourses of development are not sufficient it is proposed to be determine on basis of the required defect density DD_{result} and the latent defect density estimation $LDDE^* = \frac{N_{def}^*}{KLOC} \cdot N_{def}^*$ (calculated according to (7)). The amount of discovered defects is calculated as $(LDDE^* - \boldsymbol{D}_{result}) \cdot KLOC$. $N_{def}^{*\text{int}_k}$ is calculated according to (6). After that there is a choice n complexity intervals $(n < m)$, when the estimation of defects amount in them is $\sum_{k=1}^{n} N_{def}^{*\text{int}_k} \geq (LDDE^* - DD_{result}) \cdot KLOC$. Testing of all the components from chosen complexity intervals allows achieving DD_{result}. The permissible defects amount which was not discovered is $N_{def}^* - \sum_{k=1}^{n} N_{def}^{*\text{int}_k}$. These considerations allow formalizing a criterion of the necessary reliability achievement when the development recourses are not sufficient

$$LDDE^* = \frac{N_{def}^* - \sum_{k=1}^{n} N_{def}^{*\text{int}_k}}{KLOC}; \quad LDDE^* \leq DD_{result} \quad (9)$$

The received final results allow the developers, test – managers and testers to make decisions directed to increase the software reliability.

Stage 4. To make decisions directed to increase the software reliability.

a. The probability of the component containing defects $P_{\text{mod}\_def}^{\text{int}_k}$ according to the formulas (1) informs the testers about the advisability of testing. For example, in 10 components of one complexity interval, five components had defects. If $P_{\text{mod}\_def}^{\text{int}_k} = 0.5$, it is possible to make a reasonable conclusion, that the testing of the rest five components is not advisable.

b. The probability of components with 1, 2, 3, *i*… defects $P_{\text{mod}\_i\_def}^{\text{int}_k}$ according to the formulas (2) informs the testers about the most probable defects amount in then testing component.

c. The index of the average number of defects in the component $N_{def}^{\text{int}_k}$ according to the formulas (3) allows the developers to choose refactoring components, and the testers to sort testing components in descending order of defects amount. It facilitates the purposeful testing the components containing a great number of defects, the most rapid discovering and elimination of defects which increase the software reliability.

d. The estimation of the number of the components containing defects $N_{\text{mod}\_def}^{*\text{int}_k}$ according to the formulas (4), $N_{\text{mod}\_def}^*$ according to the formulas (5) allows test-managers and testers to determine the number of the residual components containing defects. Then it is necessary to analyze the risk of defects and to make an additional test of those components which contain critical and serious defects.

e. The number of defects estimation $N_{def}^{*\text{int}_k}$ according to the formulas (6), N_{def}^* according to the formulas (7) in the testable components allows to test-managers and testers to determine the required recourses using the average time to discover and eliminate one defect. And also to calculate the undiscovered defects number, to estimate the effectiveness of testing, to make a reasonable decision to continue or finish testing process.

f. The code base defectiveness degree *CBDD* according to the formulas (8) informs the developers about the quality of the code development and test-managers about necessary testing volumes. The following gradation of this index is offered: 1) when $CBDD \leq 30\%$ the code base defectiveness degree is low, the development quality is high it is demanded a little testing recourse volume; 2) when $30 < CBDD \leq 60\%$ we have the average defectiveness degree and we have the average development quality so the average testing recourse volume is demanded; 3) when $CBDD > 60\%$ the defectiveness degree is high and the development quality is low. Is this case the defects are in the majority of components regardless of their complexity. That is why it is necessary to test the majority of components what calls for increasing the testing recourse volume.

g. The criterion of the necessary reliability achievement according to the formulas (9) allows to choose the components for testing to achieve the demanded defects density when the development recourses are not sufficient.

3.3 The example of the method's use

The project data http://code.google.com/p/promisedata/wiki/xalan, version 2.6 (http://promisedata.googlecode.com, 2014) were used as the example for this method utilization. The project contains 885 components, more than 400 thousand lines of code (KLOC). In data's depositary the metrics' account was presented and also the defects amount for the project's components.

Stage 1. The complexity level of the software components was made. For each project's component the composite metric was calculated. The numeral diapason of the composite metric was divided into following intervals [0-2], [2-7], [7-11], [11-15], [15-20], [20-30], [30-50], [50-100], [100-130], [130-600]. 135 components of basic subset A_{base} were chosen. Each complexity interval was presented no less than 10 components with different defects amount.

Stage 2. The count of initial data was made which is given in Table 1 below.

Table 1
The Initial Data of the Basic Subset A_{base}.

№	The intervals of the composite metric	The number of components and defects							$NP^{\text{int}_k}_{\text{mod}}$ $\mathbf{A_{pred}}$
		$NB^{\text{int}_k}_{\text{mod}}$	$NB^{\text{int}_k}_{\text{mod\_}def}$	$NB^{\text{int}_k}_{\text{mod\_}i\_def}$				$NB^{\text{int}_k}_{def}$	
				with 1 def.	with 2 def.	with 3 def.	with 4 def.		
1	2	9	2	2				2	19
2	7	11	3	3				3	87
3	11	17	5	4	1			6	48
4	15	10	4	4				4	45
5	20	10	4	3	1			5	67
6	30	12	4	2	2			6	108
7	50	16	5	4	1			6	102
8	100	27	15	11	2	2		21	172
9	130	11	11	2	7	1	1	25	33
10	600	12	12	6	4	2		20	69
Total		135	65	41	17	5	1	98	750

Stage 3. The count of reliability indices was made, according to the formulas (1)-(9). The indices are given in Tables 2, 3, 4.

Table 2
The Indices S of the Components' Reliability

№	The intervals of the composite metric	Probabilities					
		$P^{\text{int}_k}_{\text{mod\_}def}$	$P^{\text{int}_k}_{\text{mod\_}i\_def}$				$\overline{N^{\text{int}_k}_{def}}$
			with 1 def.	with 2 def.	with3 def.	with 4 def.	
1	2	0,22	0,22				0,22
2	7	0,27	0,27				0,27
3	11	0,29	0,24	0,06			0,35
4	15	0,40	0,40				0,40
5	20	0,40	0,30	0,10			0,50
6	30	0,33	0,17	0,17			0,50
7	50	0,31	0,25	0,06			0,38
8	100	0,56	0,41	0,07	0,07		0,78
9	130	**1,00**	0,18	0,64	0,09	0,09	2,27
10	600	**1,00**	0,50	0,33	0,17		1,67

Table 3
The Estimation of the Components Containing Defects in A_{base}.

№	The intervals of the composite metric	The estimation of the components containing defects				
		$N^{*\text{int}_k}_{\text{mod\_}def}$	$N^{*\text{int}_k}_{\text{mod\_}i\_def}$			
			with 1 def.	with 2 def.	with 3 def.	with 4 def.
1	2	4	4			
2	7	23	23			
3	11	15	12	3		
4	15	18	18			
5	20	27	20	7		
6	30	36	18	18		
7	50	32	26	6		
8	100	96	71	12	12	
9	130	33	6	21	3	3
10	600	69	35	23	12	
	$N^{*}_{\text{mod\_}def}$	353	233	90	27	3

Table 4
The Estimation of Defects Number in Components A_{pred} .

№	The intervals of the composite metric	The estimation of defects number in components				
		with 1 def.	with 2 def.	with 3 def.	with 4 def.	$N^{*\text{int}_k}_{def}$
1	2	4				4
2	7	23				23
3	11	12	6			18
4	15	18				18
5	20	20	14			34
6	30	18	36			54
7	50	26	12			38
8	100	71	24	36		131
9	130	6	42	9	18	75
10	600	35	46	36		117
	N^{*}_{def}	233	180	81	18	512

Stage 4. Making decisions which are directed to increase Software reliability.

4.1 Table 2 represents the probability of availability of the defects containing component $P^{\text{int}_k}_{\text{mod}\_def}=1$ for the 9-th and 10-th complexity intervals. It informs the testers about the necessity of testing all the components in these intervals. The rest of other probable cases represent different advisability of testing components.

4.2 Table 2 represents the number of defects which is more probable in these components with different complexity level.

4.3 Table 2 represents that $\overline{N^{\text{int}_k}_{def}} \geq 0.5$ is for 5 complexity intervals. For the 9th complexity interval is $\overline{N^{\text{int}_k}_{def}} > 2$. This information allows developers to choose the components for refactoring and testers to sort out the components in descending order of the defects number. It makes for choosing and dedicated testing the components containing the greater number of defects and also to reveal and correct the great number of defects and to increase the software reliability

4.4 Table 3 represents that $N^{*}_{\text{mod}\_def}=353$. After testing 300 components contained defects. It means that there are as far back as 53 components with defects. It is necessary to analyze the defects' risk and additional testing the components with makes contain critical and serious defects.

4.5 Table 4 represents that $N^{*}_{def}=512$. If the average period of time for one defect reveal in the previous projects was 3 labor hours it is necessary to spend 512.3=1536 labor hours for all defects reveal. 450 defects were revealed in the conditions of limited testing recourses. The number of not revealed defects is 512-450=62. The degree of defects reveal is 450/512=0,88. To make a decision of continuing or stopping the testing is a compromise between demined software reliability and limited testing recourses. The number of unrevealed defects represents the information of maintenance organization and determination the release date of the software service pack.

4.6 Table 1 represents that the total number of components for testing is 750. The index *CBDD*=353/750 · 100%=48% shows the average quality of development code that demands the average testing volumes.

4.7 For the under test software project is *KLOD*=400, $N^{*}_{def}=512$, $LDDE^{*}=512/400=1.3$. To achieve DD_{result}=0.5 it is necessary to reveal (1.3-0.5) · 400=280 defects. From 10 complexity intervals which are represented in table 4, the intervals $INT_{select}=\{8,9,10\}$ were chosen. For these intervals is $\sum_{k=8}^{10} EN^{\text{int}_k}_{def}=323$, 323>280. After testing all the components of chosen intervals was achieved the necessary defects density and needed reliability. The criterion of the necessary reliability achievement when the development recourses are not sufficient is

$$LDDE^{*}=\frac{512-\sum_{k=8}^{10} N^{*\text{int}_k}_{def}}{400}=\frac{512-323}{400}=\frac{189}{400}=0.47;\ LDDE^{*}<0.5$$

However, it is necessary to analyze the defects' risks and those components additional testing which contain critical and serious defects.

CONCLUSIONS

The following article deals with the software reliability increase method which allows to calculate the following indices: the probability of existence of the component containing defects; the probability of existence in components 1,2,3,*i*... defects; the average number of defects in the components; the estimation of the component containing defects in the complexity

intervals; the estimation of the amount of the components containing defects in the predictive subset; the estimation of the amount of defects in the complexity intervals; the estimation of defects amount in the predictive subset; the code base defectiveness degree; the criterion of the necessary reliability achievement when the recourses of development are not sufficient.

The verification of the method was made for twenty Software projects. The results showed, that the average estimations deflection of real amount of components containing defects were 3,22%; the estimations of component with 1,2,3,4 defects were 5,08%; the estimations of the total defects amount were 7,83%. Insignificant (till 10%) deflections prove that the offered method makes the required accuracy of quantitative estimation of reliability indices.

The demanded data set for the following method obtain by means of the counting the components containing defects and defects in tested part of developing program project.

It provides the accounting of its specific complexity and development peculiarities. It facilitates increasing of estimation accuracy. The accuracy and exactness of getting indices are undoubted advantages of the method.

In contrast to the basics Bayesian belief networks the given method does not use the expert estimations. In contrast to the basics of neural networks and the machine learning the given method has the simple mathematics calculations which may be made by means of electron tables' editor. This method does not use the complex and expensive program tools.

For automatization the initial data set accounting it is necessary to use the simple program tool which can be created by any software company specialists. The specialty of the method is the static multidimensional complexity accounting which is based on well-known standards of metrics. Therefore, the method is independent from the programming language.

This method is used in the software development by different companies in Izmail, Ukraine. This method allowed reducing the unrevealed defects number, to decrease the estimation of the latent defects density, to increase the software reliability on the average 8 %. The practical application of the method confirmed the possibility and advisability of its use in the engineering program practice to increase the reliability. It is especially actual when the recourses of development are not sufficient.

REFERENCES

ISO/IEC 25010. (2011). *Systems and Software engineering - Systems and software quality requirements and evaluations (SQuaRE) - System and Software Quality models*.

Maevsky, D. A., & Yaremchuk, S. A. (2012). A priori estimation of the amount of faults in information system software. *Radio Electronic and Computer Systems, 4*(56), 73-80. Kharkiv: KHAI,.

Maevsky, D. A., & Yaremchuk, S. A. (2012). The estimation of the amount of software faults on the complexity metric basis. *Electrical Engineering and Computer Systems, 07*(83), 113–120. Kiev: Technica.

Neumann, P. G. (1995). Computer related risks. *Reading*. MA: Addison-Wesley.

IEEE Std 610.12. (1990). *IEEE standard glossary of software engineering terminology*.

Ma Y., Guo L., Cukic B. (2007). Statistical framework for the prediction of fault proneness. *Advances in machine learning application in software engineering* (pp.237–265). Idea Group Inc..

Mahaweerawat, A., Sophasathit, P., & Lursinsap, C. (2002). *Software fault prediction using fuzzy clustering and radial basis function network. In International conference on intelligent technologies. Vietnam, 304-313.*

Thwin, M. M. T., & Quah, T.-S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. *J. System Software. May., 76*, 147–156.

Pomorova, O. V., & Hovorushchenko, T. O. (2012). The research of Mat Lab function features for scaling input data of Software quality evaluation artificial neural network. *Radio Electronic and Computer Systems, 5*(57). Kharkiv: KHAI, 219-224.

Fenton, N. E., & Neil, M. A. (1999). Critique of software defect prediction models. IEEE Trans. *Softw. Eng., 25*(5), 675–689.

The PROMISE Repository of empirical software engineering data. http://promisedata.googlecode.com – 01-04-2014.

