РАДІОТЕХНІКА, ЕЛЕКТРОНІКА ТА ТЕЛЕКОМУНІКАЦІЇ

UDC 681.3

VERA YURIIVNA TITOVA
Khmelnytsky National University

COMPARATIVE ANALYSIS OF NEURAL NETWORKS FOR THE EVALUATION OF SOFTWARE QUALITY

The problem of software quality evaluation is considered in this paper. Software quality is evaluated using a quality model. The quality model consists of software quality metrics classified into a hierarchical tree structure. The upper level of this structure consists of quality characteristics, and the lower level consists of software quality attributes. Based on the analysis of these characteristics and attributes, the authors determine that current quality model is not formalized. So, they propose the formalized model of software quality. This model is the basis for an unsupervised neural network for software quality evaluation. Based on the comparative analysis of clustering validation indexes a Kohonen SOM is chosen. The model and the neural network developed in this paper become the basis for developing a software quality evaluation system.

Keywords: *Kohonen SOM, ART-network, Clustering Validation Indexes, Software Quality Model, Neural Networks, Unsupervised Learning, Comparative Analysis.*

В.Ю. ТІТОВА
Хмельницький національний університет

ПОРІВНЯЛЬНИЙ АНАЛІЗ НЕЙРОННИХ МЕРЕЖ ДЛЯ ОЦІНКИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В роботі розглядається проблема оцінки якості програмного забезпечення. Якість програмного забезпечення оцінюється за допомогою моделі якості. Модель якості складається з показників якості програмного забезпечення, класифікованих у ієрархічну структуру дерева. Верхній рівень цієї структури складається з характеристик якості, а нижній рівень складається з атрибутів якості програмного забезпечення. На підставі аналізу цих характеристик та атрибутів автори визначають, що поточна модель якості є неформалізованою. А тому, вони пропонують формалізовану модель якості програмного забезпечення. Зазначена модель є основою нейронної мережі, що навчається без вчителя, для оцінки якості програмного забезпечення. На основі порівняльного аналізу показників кластеризації авторами обрано нейронну мережу Кохонена. Модель і нейронна мережа, розроблені в даній статті, є наступним кроком для розробки системи оцінки якості програмного забезпечення.

Ключові слова: АРТ-мережа, індекси перевірки кластеризації, модель якості програмного забезпечення, нейронні мережі, неконтрольоване навчання, порівняльний аналіз.

Introduction. Improving of software quality is one of the important and actual tasks of software development. The solution of this problem is especially important for critical software, which is related to the safety of people.

There is no single approach to software quality evaluation for today. So, the development of regulatory framework that defines software quality requirements and the development of methods for the evaluating implementation of these requirements are needed for software quality improving [1].

One of the ways for the evaluation of software quality is evaluation using a quality model. There are several software quality models for today. Their comparative analysis is presented in this study [1]. Based on it, we can conclude that the software quality model described here [2] is the most relevant for software development.

This quality model consists of two parts [2]: a product quality model and a quality in use model. The characteristics and subcharacteristics of these two models are shown in Fig. 1 [2]:

These characteristics and subcharacteristics are the input data for the evaluation of software quality. They are divided into [2]:

- descriptive, that describe the set of tools and general properties of an object, its functions, security and importance;
- quantitative, which can be measured and numerically compared with requirements;
- qualitative, which are determined by expert method.

So, the using of mathematical methods to solve the problem of software quality evaluation is impossible because the input data is heterogeneity. The formalization of the software quality model enables to simplify the solution of this problem and, as a result, to improve the quality of using software.

Formalized model of software quality. For the formalization of the model we use sets theory apparatus. We mark total quality - Q. It's calculated by the values of product quality characteristics Q_P and by the values of quality in use characteristics Q_U. Q_P is determined from the sets of characteristics:

- functional suitability FS, $FS = \{fs_1, fs_2, fs_3\}$, where fs_1 - functional completeness, fs_2 - functional correctness, fs_3 - functional appropriateness;
- performance efficiency PE, $PE = \{pe_1, pe_2, pe_3\}$, where pe_1 - time behavior, pe_2 - resource utilization, pe_3 - capacity;

Product Quality Model

(Sub)Characteristic	(Sub)Characteristic
Functional suitability	**Reliability**
Functional completeness	Maturity
Functional correctness	Availability
Functional appropriateness	Fault tolerance
Performance efficiency	Recoverability
Time behaviour	**Security**
Resource utilization	Confidentiality
Capacity	Integrity
Compatibility	Non-repudiation
Co-existence	Accountability
Interoperability	Authenticity
Usability	**Maintainability**
Appropriateness recognizability	Modularity
Learnability	Reusability
Operability	Analysability
User error protection	Modifiability
User interface aesthetics	Testability
Accessibility	**Portability**
	Adaptability
	Installability
	Replaceability

Quality in use Model

(Sub)Characteristic
Effectiveness
Efficiency
Satisfaction
Usefulness
Trust
Pleasure
Comfort
Freedom from risk
Economic risk mitigation
Health and safety risk mitigation
Environmental risk mitigation
Context coverage
Context completeness
Flexibility

Fig. 1. Quality model characteristics and subcharacteristics

- compatibility C, $C=\{c_1,c_2\}$, where c_1 - co-existence, c_2 - interoperability;
- usability U, $U=\{u_1,u_2,u_3,u_4,u_5,u_6\}$, where u_1 - appropriateness recognizability, u_2 - learnability, u_3 - operability, u_4 - user error protection, u_5 - user interface aesthetics, u_6 - accessibility;
- reliability R, $R=\{r_1,r_2,r_3,r_4\}$, where r_1 - maturity, r_2 - availability, r_3 - fault tolerance, r_4 - recoverability;
- security S, $S=\{s_1,s_2,s_3,s_4,s_5\}$, where s_1 - modularity, s_2 - confidentiality, s_3 - integrity, s_4 - modifiability, s_5 - testability;
- maintainability M, $M=\{m_1,m_2,m_3,m_4,m_5\}$, where m_1 - modularity, m_2 - reusability, m_3 - non-repudiation, m_4 - accountability, m_5 - authenticity;
- portability P, $P=\{p_1,p_2,p_3\}$, where p_1 - adaptability, p_2 - installability, p_3 - replaceability.

From the definitions of characteristics we can conclude that they are interrelated. For example, usability depends on performance efficiency, and reliability depends on maintainability. So, to determine product quality, we use a multiplicative index or the product of sets.

$$Q_p = FS \times PE \times C \times U \times R \times S \times M \times P \quad (1)$$

The quality of use consists of characteristics or the sets of characteristics: effectiveness - es; efficiency - ey; satisfaction - ST, $ST=\{st_1,st_2,st_3,st_4\}$, where st_1 - usefulness, st_2 - trust, st_3 - pleasure, st_4 - comfort; freedom from risk - FR, $FR=\{fr_1,fr_2,fr_3\}$, where fr_1 - economic risk mitigation, fr_2 - health and safety risk mitigation, fr_3 - environmental risk mitigation; context coverage - CC, $CC=\{cc_1,cc_2\}$, where cc_1 - context completeness, cc_2 - flexibility.

These characteristics are dependent on the characteristics of product quality. For example, effectiveness depends on the characteristics of functionality suitability, reliability, usability, maintainability, portability; satisfaction - on the characteristics of functionality suitability, portability and usability.

So, to evaluate total quality, we use the multiplicative index again.

$$Q = Q_P \times Q_U \quad (2)$$

We can present the formalized software quality model in the following form:

$$Q = (FS \times PE \times C \times U \times R \times S \times M \times P) \times Q_U \quad (3)$$

A similar model has already been considered in [3], but it has the following drawbacks. Firstly, it doesn't take into consideration the relationships between the product quality parameters. Secondly, it does not take into consideration the type of software. The latter is important, since different quality parameters can be important for the different types of software.

We determine the following types of software based on software classification [4]:

- critical software or the software of high importance - is the software that performs critical functions that are important to security, that is, software whose failure to perform functions or its misuse or negligence can become catastrophic or critical consequences. Automated systems in the space industry, the nuclear industry,

medicine and other spheres are the example of such software;

- the software of medium importance – is the software whose failure to perform functions or its misuse or negligence can become financial or information losses, but not catastrophic or critical consequences. System software and some application programs are the example of such software;
- the software of low importance - is the software whose failure to perform functions or its misuse or negligence can become the moral dissatisfaction of users and haven't other consequences. Computer games and other entertainment programs are the example of such software.

So, we have the set of software types $CL = \{cl_1, cl_2, cl_3\}$, where cl_1 - critical software, cl_2 - the software of medium importance, cl_3 - the software of low importance. Given relationships between quality characteristics and software types, we have the following quality model (4).

$$Q = ((FS \times PE \times C \times U \times R \times S \times M \times P) \times Q_U) \times CL \tag{4}$$

We replace sets by their corresponding characteristics and subcharacteristics, and we obtain the following formula (5), which is the Formalized Model of software quality.

We can conclude from the analysis of the model, that:

- software quality depends on the large number of interrelated characteristics;
- the evaluation of software quality cannot be reduced to usual numerical calculations.

$$Q = \left\{ \begin{array}{l} (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_1, cl_1), \\ (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_1, cl_2), \\ (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_1, cl_3), \\ (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_2, cl_1), \\ (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_2, cl_2), \\ (fs_1, pe_1, c_1, u_1, r_1, s_1, m_1, p_1, es, ey, st_1, fr_1, cc_2, cl_3), \\ \vdots \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_1, cl_1), \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_1, cl_2), \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_1, cl_3), \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_2, cl_1), \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_2, cl_2), \\ (fs_3, pe_3, c_2, u_6, r_4, s_5, m_5, p_3, es, ey, st_4, fr_3, cc_2, cl_3), \end{array} \right\} \tag{5}$$

So, the problem of software quality evaluation belongs to the difficult formalized tasks. Today, artificial neural networks are one of the perspective ways to solve such problems. So, we use a neural network method to evaluate software quality.

Analysis of neural networks for software quality evaluation. One of the neural networks for software quality evaluation is considered in this paper [3]. It belongs to the supervised neural networks.

However, in the process of training this neural network, difficulties arise with the accuracy of output training sets. Since the data obtained through expert evaluation can be incorrect due to the subjectivity of experts. For example, different experts from developers, users or customers can differently evaluate software quality for the same input data. And the data obtained through the testing of software matching with requirements for it can be incorrect, because the test results are determined by the values "passed" and "not passed," without specifying the degree of passing. So, we decide to use an unsupervised neural network to avoid difficulties in the formation of training sets.

Among the most perspective unsupervised neural networks for today are an ART-2 network and a Kohonen SOM. So, we compare these two networks based on a training data set. The training data set is the values of quality characteristics and subcharacteristics described above. The size of the training set is 150 that match to 150 tested programs. The values of the input data are determined using the method described here [5]. Software testing takes place on the base of software of the Khmelnytsky National University information and computer center.

For the neural network comparative analysis we use the following clustering validation indexes [6–8]:

- Davies–Bouldin Index (DBI) - is a function of the ratio of sum of within-cluster scatter to between-cluster separation. The ideal DBI presents minimal ratio of within-cluster scatter and between-cluster separation; therefore, minimizing within-cluster scatter and maximizing between-cluster separation are desired;
- Calinski–Harabasz Index (CHI) - is a function of the ratio of sum of squares among the clusters to sum of squares within the clusters. A better clustering result is indicated by a higher CH value;
- Ray–Turi Index (RTI) - is a function of the ratio of the intra-cluster distance to minimal of inter-cluster distance. The clustering result which gives a minimum RTI tells us what the ideal number of clusters is since minimizing inter-cluster distance and maximizing inter-cluster one are presented;
- Dunn Index (DI) - is a function which takes the minimal ratio of inter-cluster distance to maximal intra-cluster distance. The main goal of DI is to maximize inter-cluster distances and minimize intra-cluster distances. Therefore, the number of clusters that maximizes DI is taken as the ideal clustering result.

The number of output clusters can be two, three, four, five or six.

If the number of clusters is two, then they take the following values: software needs rework and software doesn't need rework.

If the number of clusters is three, then they take the following values: software needs full rework; software needs rework, but not a full one; software doesn't need any rework.

If the number of clusters is four, then they take the following values: software needs full rework; software

needs large rework; software needs little rework; software doesn't need any rework.

If the number of clusters is five, then they take the following values: software needs full rework; software needs large rework; software needs medium rework; software needs little rework; software doesn't need rework.

If the number of clusters is six, then they take the following values: software needs full rework; software needs large rework; software rather needs large rework than little rework; software rather needs little rework than large rework; software needs little rework; software doesn't need rework.

We implement both neural networks in Matlab. The results of networks comparison are shown in Fig. 2.

We can conclude from this comparison that the Kohonen SOM shows the best results for all indexes. Therefore, we choose this neural network for software quality evaluation.

The optimal clusters number is also selected from the results of comparison. Three indexes have the maximum value for four clusters. The value of the fourth index is satisfactory for this clusters number. So, we evaluate software quality by four classes: software needs full rework; software needs large rework; software needs little rework; software doesn't need any rework.

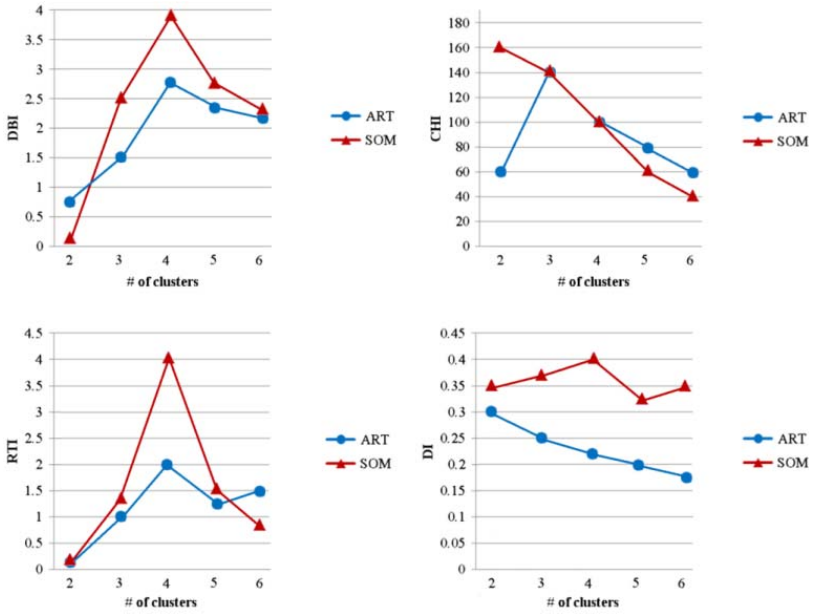

Fig. 2. The results of networks comparison

Results of experiments. We test selected neural network for software quality evaluation in the training set. The fragment of this set is shown in Table 1.

Table 1

Training fragment set for the neural network for software quality evaluation

	Soft. No.1	Soft. No.2	Soft. No.3	Soft. No.4	Soft. No.5	Soft. No.6	Soft. No.7	Soft. No.8	Soft. No.9	Soft. No.10
fc_1	1	1	0.9	0.9	0.8	0.8	0.9	1	0.9	1
fc_2	1	1	0.9	0.9	0.8	0.8	0.9	1	0.9	1
fc_3	1	1	0.9	0.9	0.8	0.8	0.9	1	0.9	1
pe_1	1	1	1	1	0.9	0.9	1	1	1	1
pe_2	1	1	1	1	0.9	0.9	1	1	1	1
pe_3	1	1	1	1	0.9	0.9	1	1	1	1
c_1	1	1	1	1	1	1	1	1	1	1
c_2	1	1	1	1	1	1	1	1	1	1
u_1	1	1	1	1	1	1	1	1	1	1
u_2	1	1	1	1	1	1	1	1	1	1
u_3	1	1	1	1	1	1	1	1	1	1
u_4	1	0.9	0.85	0.8	1	1	0.8	1	1	1
u_5	1	1	1	0.9	1	1	0.9	1	1	1
u_6	1	1	1	1	1	1	1	1	1	1
r_1	1	1	1	1	1	1	1	1	1	1
r_2	1	1	1	1	0.8	1	1	1	1	1
r_3	1	0.9	0.9	0.8	07	0.9	0.8	1	1	1
r_4	1	1	1	0.9	0.7	0.9	0.9	1	1	1
s_1	1	1	1	1	1	1	1	1	1	1
s_2	1	1	1	1	1	1	1	1	1	1

Table 1 (continue)

	Soft. No.1	**Soft. No.2**	**Soft. No.3**	**Soft. No.4**	**Soft. No.5**	**Soft. No.6**	**Soft. No.7**	**Soft. No.8**	**Soft. No.9**	**Soft. No.10**
s_3	1	1	1	1	1	1	1	1	1	1
s_4	1	1	1	1	1	1	1	1	1	1
s_5	1	1	1	1	1	1	1	1	1	1
m_1	1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1	1
m_2	1	1	1	1	1	1	1	0.9	1	1
m_3	1	1	1	1	1	1	1	0.9	1	1
m_4	1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1	1
m_5	1	1	1	1	1	1	1	0.9	1	1
p_1	1	0.9	0.9	0.9	0.9	0.9	0.9	1	1	1
p_2	1	1	1	1	1	1	1	1	1	1
p_3	1	0.9	0.9	0.9	0.9	0.9	0.9	1	1	1
es	1	1	1	1	0.75	1	1	1	1	1
ey	1	1	1	1	0.6	1	1	1	0.9	0.9
st_1	1	1	1	1	1	1	1	1	0.8	0.95
st_2	1	1	1	1	1	1	1	1	0.8	0.95
st_3	1	1	1	1	1	1	1	1	0.8	0.95
st_4	1	1	1	1	1	1	1	1	0.8	0.95
fr_1	1	1	1	1	1	1	1	1	1	1
fr_2	1	1	1	1	1	1	1	1	1	1
fr_3	1	1	1	1	1	1	1	1	1	1
cc_1	1	1	1	1	0.8	1	1	1	1	1
cc_2	1	1	1	0.9	0.6	0.9	0.9	1	1	1
cl_1	0	0	0	0	1	0	0	0	0	0
cl_2	1	0	1	0	0	1	1	0	0	0
cl_3	0	1	0	1	0	0	0	1	1	1

Input values consist of quality subcharacteristics and software classes. The values are in the range [0..1]. A better subcharacteristics result is indicated by a higher value. The software classes contain 1 in the corresponding class and 0 in the other classes. The set size is 10. The results of neural network work are shown in Fig. 3.

Red cluster corresponds to software that needs full rework. Software *No.5* matches this class. Yellow cluster corresponds to software that needs large rework. Software *No.6* and *No.7* match this class. Blue cluster corresponds to software that needs little rework. Software *No.3*, *No.4* and *No.9* match this class. Green cluster corresponds to software that doesn't need any rework. Software *No.1*, *No.2*, *No.8*, and *No.10* match this class.

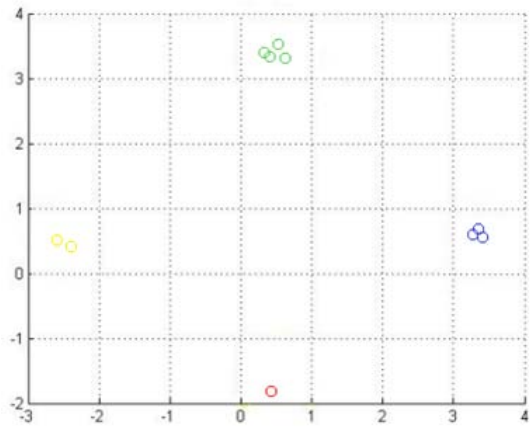

Fig. 3. Results of clustering

At the similar training set, we test a supervised neural network based on formalized quality model described here [3]. The test results are as follows:

- $Q_1 = 1$, $Q_2 = 0.98$, $Q_3 = 0.95$, $Q_8 = 0.98$, $Q_{10} = 0.95$. These values correspond to 100% quality, 98% quality, 95% quality, respectively, and they mean that software doesn't need rework;
- $Q_4 = 0.88$, $Q_6 = 0.85$, $Q_7 = 0.85$, $Q_9 = 0.88$. These values correspond to 88% quality and 85% quality, and they mean that software needs little rework;
- $Q_5 = 0.65$. This value corresponds to 65% quality. It means that software needs large rework.

These results show us that:

- the evaluation of software quality without considering the relationship between the subcharacteristics of the quality model compensates for the worst values of some subcharacteristics by the best values of others. So, the total quality value is overestimated;
- the evaluation of software quality without considering software class doesn't takes into account the importance of quality model subcharacteristics. As a result, the total quality value is overestimated again.

So, we can conclude that the formalized software quality model described in this study is more relevant for the problem of software quality evaluation.

Conclusions and future work. In this study we analyze and formalize conditions that characterize the software quality model. This lets us to develop the Formalized Model of software quality.

The analysis of this model shows that to solve the problem of software quality evaluation it's better to use unsupervised neural networks. An ART-2 network and a Kohonen SOM are the most perspective networks of this type for today.

For the analysis of these networks we use the training set that consists of the values of quality characteristics and subcharacteristics. The size of this training set is 150 that match to 150 tested programs.

Based on the comparative analysis of these two networks, we conclude that the Kohonen SOM better suits for solving the problem of software quality evaluation. So, we choose this neural network to solve this problem.

Developed model and neural network are the basis for developing a software quality evaluation system. We expect that the using of this system automates a software quality evaluation process. As a result, this enables to avoid subjectivity when evaluating software quality, to improve software quality and to make the software quality evaluation process more economically profitable.

References

A Review of Software Quality Models for the Evaluation of Software Products / José P. Miguel, David Mauricio, Glen Rodríguez. // International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, November 2014.

ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

Neiromerezhnyi metod dlia vyznachennia yakosti prohramnoho zabezpechennia krytychnoho zastosuvannia / V.Iu. Titova // Shtuchnyi intelekt. – 2012. – № 4. – P. 594–601.

ISO/IEC 26514:2008 - Systems and software engineering - Requirements for designers and developers of user documentation.

Nechitka neironna merezha dlia vyznachennia vidpovidnosti rezultativ testuvannia prohramnoho zabezpechennia krytychnoho zastosuvannia vymoham / V.Iu. Titova // Shtuchnyi intelekt. – 2013. – № 4. – P. 548–554.

Cheng-Ching Chang, Ssu-Han Chen. A comparative analysis on artificial neural network-based two-stage clustering. // Cogent Engineering. – vol. 2, 2015. - Issue 1. - https://www.tandfonline.com/doi/full/10.1080/23311916.2014.995785

On the number of clusters in block clustering algorithms / M. Charrad, Y. Lechevallier, M. B. Ahmed, G. Saporta. // Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference, 2010. – P. 392–397.

A new distance measurement for clustering time-course gene expression data / G. Chen, Y. Dai. // Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco, CA, 2004. – P. 2929–2932.

Рецензія/Peer review : 29.9.2018 р. Надрукована/Printed : 18.9.2018 р.
Рецензент: д.т.н., проф. Боровик О.В.