An Approach to Early Prediction of Software Quality*

YAO Lan¹, YANG Bo²

¹. School of Electronic Engineering, University of Electronic Science and Technology of China
Chengdu 610054 China
². School of Mechatronics Engineering, University of Electronic Science and Technology of China
Chengdu 610054 China

Abstract Due to the rapid development of computers and their applications, early software quality prediction in software industry becomes more and more crucial. Software quality prediction model is very helpful for decision-makings such as the allocation of resource in module verification and validation. Nevertheless, due to the complicated situations of software development process in the early stage, the applicability and accuracy of these models are still under research. In this paper, a software quality prediction model based on a fuzzy neural network is presented, which takes into account both the internal factors and external factors of software. With hybrid-learning algorithm, the proposed model can deal with multiple forms of data as well as incomplete information, which helps identify design errors early and avoid expensive rework.

Key Words early quality prediction; fuzzy neural network; prediction model; software internal attributes; software quality attributes

With the rapid development of computer hardware, the bottleneck of system quality lies in its software. Therefore, people are looking forward to reaching such a situation where software works with high reliability, maintainability and performance, short development time, and low development and maintenance cost.

One approach to solving the above problem effectively is to predict quality of software in the early stages of the software development process. For an established software company, there are often some data collected from past projects or releases. They are useful for early prediction of software quality. Early prediction, i.e., predicting the quality software modules prior to software testing and operations, benefits software development teams from many aspects. On the one hand, it will be possible to estimate the impact that will be exerted on the final software product, and such timely quality estimation can be used to direct cost-effective quality improvement efforts to the high-risk modules, so that the quality of software can be controlled and guaranteed by developers in advance. On the other hand, it is helpful to find the “best” software design out of numerous options, namely to find the design that will create the highest software quality among a large number of feasible software designs sharing the common function of customers’ requirements but with different software non-functional attributes which results in different software quality. Above all, it is obvious that this approach of predicting the quality of software in the early stages of the development process does make sense in ensuring the final quality of software product, shortening the development cycle and reducing the cost of development and maintenance. Developers and customers can come up with realistic overall requirements for the target system early, avoiding possible expensive rework in later stages of the software lifecycle.

Nowadays, most software quality prediction models use classic statistical methods, classification and clustering algorithms, etc, or take a multivariate approach. Zhang et al. used a Bayesian belief network (BBN) based approach to capture design rationale as correlations among the design decisions and quality attributes, which helps developers make rationale designs when building a specific system[1]. Khoshgoftaar et al. presented a case study from comprehensive evaluation (with several large case studies) of currently available regression tree (CART) algorithms: CART-LS (least squares), S-plus, and CART-LAD (least absolute deviation) for software fault prediction[2]. Furthermore Khoshgoftaar presented an in-depth study of calibrating classification
trees for software quality estimation using the serial parallelizable induction of decision trees (SPRINT) algorithm, which removes all of memory limitations including the requirement that data sets be memory resident and provides a fast and scalable analysis[3]. Yuan et al. presented a modeling technique that integrated fuzzy subtractive clustering with module-order modeling for software quality prediction to identify whether modules are fault-prone or not[4]. Wang et al. introduced an interpretable neural network model for software quality prediction by extracting comprehensible rules from the trained neural network[5].

Nevertheless, current software quality prediction models still could not describe the complex uncertain relationship between software quality external factors and internal factors effectively, and these models often have restricted capability in the case of only incomplete information available, which is quite often in the early stages of software development process, and in the case of concurrence of multiple types of sample data such as qualitative and quantitative data.

In this paper, we propose an approach to software quality prediction based on fuzzy neural network. Artificial neural network is well-known for its abilities of learning, self-adaptation, distributed storage, and paralleled processing, and can reflect complex uncertain relationships between inputs and outputs effectively as a black box. Therefore, it is fit for describing and analysing the complex uncertain relations between software quality and software internal attributes with incomplete information in a multiple inputs and multiple outputs (MIMO) system. At the same time, fuzzy logic is good at handling quantitative information and dealing with human knowledge. Therefore, it is reasonable to introduce it to describe the causality between multiple inputs and multiple outputs exactly and effectively in conformity with multiple forms of data.

1 Description of Software Quality Attributes and Software Internal Attributes

1.1 External Factors of Software

As is known to all, quality is an integral concept that is “hard to define, impossible to measure, easy to recognize”[6]. In order to define such a concept, a lot of research has been done. At last, a number of quality attributes (also known as quality characteristics, quality factors or quality criteria) are defined to describe quality[7-10]. The ISO 9126 standard for information technology provides a framework for the evaluation of software quality, which defines six product quality characteristics, i.e. functionality, reliability, usability, efficiency, maintainability, portability[11]. Software quality attributes are also called as non-functional requirements in software engineering context. Different clients pay particular attention to different software quality attributes for different purpose. It should be emphasized that the quality of specific software holds the balance of its success in software market. If the system can satisfy all the functional requirements, e.g., an ATM machine can issue the correct amount of money that the customer asks to withdraw, but fails to satisfy any NFR, e.g., the response time is too long, then the software system will hardly be considered as a good one and the customer is unlikely to accept the system. This issue is even more crucial for information systems.

1.2 Internal Factors of Software

Software can be seen as an entity of a series of internal attributes, such as structure, size, complexity and so on, which are capable of describing the internal structure and character of the software exactly and comprehensively to a considerable extent. For the purpose of being appropriate and comprehensive, metrics are introduced to standardize and measure the internal attributes. To date, there are large numbers of metrics have been posed[12-14], among which Chidamber & Kemerer (C&K) metrics are proved and recognized to be a typical and useful set of object-oriented (OO) software metrics, including weighted methods for per class (WMC), depth of the inheritance tree (DIT), number Of children (NOC), coupling between object classes (CBO), response for a class (RFC), and lack of cohesion in methods (LCOM)[15].

1.3 Relationship between Software Quality Attributes and Internal Attributes

Quality attributes of OO software are external measurements based on customers’ subjective feeling and experience, while OO software internal attributes are objective metrics which can be measured by strict rules and principles with instruments. It's easy to recognize that some kind of complex and uncertain causal relations among some of the software internal attributes and specific quality attributes must exist that contributes to different impact on each specific quality.
attribute exerted by different combination of software internal attributes. So if objective internal attributes of a specific OO software system are available, the responding external quality attributes can be achieved somehow. A feasible way of determining the exact relations is to establish an appropriate model, so as to address the following two issues:

1) Analyze and determine which software internal attributes have a close causal relationship with the specific quality attributes.

2) Express these causal relationships accurately.

If a prediction model with the above functions can be formed, the external quality attributes of a random software entity can be determined with certain information of internal attributes, in turn the prediction of the OO software quality is realized.

2 Prediction of Object-Oriented Software Based on Fuzzy Neural Network Model

2.1 Overview of Fuzzy Neural Network

Due to limited information that can be obtained in the early stage of the software development, the prediction model must be capable of handling incomplete information effectively. In addition, the data that a prediction model has to deal with may be of various forms, such as precise data from the history software engineering project, and fuzzy information from experts’ knowledge and experience. Therefore, the adaptability to all kinds of information and compatibility between various forms of data of the software prediction model must be enhanced.

Artificial neural network is a scientific method to simulate some structures and functions of the real human neural system with physics equipment, which is famous for its ability of learning, recognition, controlling and prediction. Fuzzy logic method is good at handling uncertain information and dealing with human concept naturally. Neural networks and fuzzy logic both represent potential candidates for dealing with complex and ill-defined systems. With fuzzy logic systems can be structured and carried out as inference mechanism under cognitive uncertainty. However, fuzzy systems do not have learning and adaptation capabilities, which are the main advantages of neural networks. Thus it is by all appearances that artificial neural network and fuzzy logic are complementary technologies. The motivation for integration of artificial neural network and fuzzy logic is to take advantage of the capabilities of neural networks and fuzzy logic so that it is possible to deal with cognitive uncertainties while guaranteeing flexibility, fast and parallel computations.

2.2 Software Prediction Model Based on Fuzzy Neural Network and Hybrid-Learning Algorithm

In conformity to the above statement, establishing a prediction model based on connectionist fuzzy neural network is a feasible approach to handle multiple forms of data and incomplete information. In this paper, a fuzzy neural network model is introduced to conduct software prediction.

![A fuzzy neural network architecture model](image)

Fig.1 A fuzzy neural network architecture model

Fig.1 is a fuzzy neural network architecture model, incorporating the principle of fuzzy self-adaptation learning control network[16]. This model consists of five layers, each of which is made up of nodes with links to let information in and out. Every arrow on each link indicates a signal flow direction when this network is in use. Nodes at layer 1 are input nodes representing input linguistic variables. Layer 5 is the output layer. Different from layer 1, there are two linguistic nodes for each output variable. One is for training data (sample output) to feed into this network with the reverse direction of signal flows only used in the self-organized learning phase, and the other one with normal direction signal flow is for decision signal (actual output) to be pumped out of this network in the following optimized learning phase. Nodes at layer 2 and layer 4 are term nodes acting as membership functions to represent the terms of the respective variables in linguistic nodes. It should be noted that the number of the term nodes for each linguistic node is not fixed but depends on the need of specific problem...
under study. Each node at layer 3 is a rule node, which represents a fuzzy logic rule with fuzzy inputs from layer 2 and fuzzy outputs to layer 4. The number of the rule nodes in layer 3 is determined by the product of the number of term nodes in layer 2 for each linguistic node. Each rule node is pointed to layer 4 nodes by several links, and each link comes from exact one of the term nodes of a linguistic node. Links between layer 2 and layer 3 are called precondition links denoting the preconditions of the fuzzy rules represented by layer 3 nodes, while links between layer 3 and layer 4 are called consequent links pointing to the consequences of the fuzzy rules. Every consequent link has an associated weight that represents the strength of the existence of the corresponding rule consequence.

Based on the above structure, we adopt a hybrid-learning algorithm to realize the process of prediction.

In the first learning phase, we adopt Kohonen’s feature-maps algorithm to find out the center of each membership functions of both input and output variables [17]. Then their widths can be determined by the n-nearest-neighbors heuristic or simply by the first-nearest-neighbors heuristic. After parameters of the membership functions have been found, signals from both external sides can reach the output points of term nodes at layer 2 and 4. The outputs of term nodes at layer 2 can be transmitted to rule nodes through the initial architecture of layer 3 links. So the firing strength of each rule node is available. Based on these firing strengths and outputs of the term nodes at layer 4, we make use of the competitive learning law to update the weights of layer 4 [18]. After the competitive learning through the whole training sample data set, the link weights at layer 4 represent the strength of the existence of the corresponding rule consequence. Among the links, which connect a rule, node that the term nodes of an output linguistic node, at most one link with maximum weight is chosen and others are deleted. If all the links between a rule node and the layer 4 nodes are deleted, then this rule node can be eliminated since it does not affect the outputs. After fuzzy logic rules have already been found, the structure of the network is established.

In the second learning phase, an optimized learning algorithm is used to optimally adjust the parameters of the membership functions. After a comparison of genetic algorithm (GA) and back propagation (BP) algorithm, we adopt genetic algorithm as our optimized learning algorithm. Parameters of the membership functions are adjusted gradually in each time of training, and the learning algorithm will complete when it meets a given requirement of root mean square error between sample outputs and actual outputs. Genetic algorithm has many advantages over the traditional Back propagation algorithm in terms of convergence and time required for model training, which has been proven by our experiments.

2.3 A Sample Study of OO Software Prediction Model

According to the previous statements, here, we chose reliability, maintainability, efficiency from 6 software quality attributes: functionality, reliability, usability, efficiency, maintainability, portability as 3 outputs, and WMC, DIT, RFC from 6 internal attributes: WMC, DIT, NOC, CBO, RFC and LCOM as 3 inputs. To initiate the learning scheme, training sample data of all inputs and outputs variables are provided by experts’ knowledge and history data.

Fig.2 illustrates the prediction model before training. It can be seen that the precondition links and consequent links are fully connected between linguistic nodes and their corresponding term nodes. This means all of the possible causal relations between inputs and outputs of this system are involved, but it does not mean all of them are necessary. In this example we choose sigmoid type function as membership function for each term node in both layer 2 and 4. And each node in layer 3 performs a fuzzy AND operation.

![Fig.2 An OO software prediction model based on fuzzy neural network before training](image)
After training with hybrid-learning algorithm, the structure and all parameters of this model are confirmed, as shown in Fig.3. Each node in layer 3 represents an independent fuzzy logic rule after training as follows.

Rule1 (R1): If WMC is Many, DIT is Deep and RFC is Few, Then Reliability is High, Maintainability is High and Efficiency is Low.

Rule2 (R2): If WMC is Few, DIT is Deep and RFC is Many, Then Reliability is High, Maintainability is Low and Efficiency is High.

Rule3 (R3): If WMC is Few, DIT is Shallow and RFC is Many, Then Reliability is Low, Maintainability is Low and Efficiency is High.

The results of the emulator show that by training and learning with the experts’ knowledge and history data, this prediction model obtains all fuzzy logic rules that really exist in a specific system, which describe the mapping relations between software internal attributes and quality attributes properly and effectively. That is to say, once the prediction model is structured and completes training, by putting given inputs into this prediction model and analyzing the outputs obtained, the influence that different combinations of the internal attributes exert on a specific quality attribute can be told, so that the final quality of a software product can be predicted. It is helpful for developers to make decisions by applying these prediction results as feedback to keep the software process under control.

2.4 Dealing with Multiple Forms of Data and Incomplete Information

Generally speaking, in the early stage of the software development information available is likely to be of multiple forms. On the one hand, sample data for training are usually precise quantitative data coming from the experience data of the same or similar software system, such as WMC is 20, RFC is 35, Reliability is 80%, etc. These data can be used directly as inputs or outputs in the prediction model. On the other hand, sample data are also possible to be fuzzy qualitative information coming from experts’ knowledge, e.g. If WMC is Many, RFC is Few, DIT is Deep, Then Reliability is High, Maintainability is High, Performance is Low. In this case, fuzzy information can be added to the prediction model as a fuzzy logic rule without training.

Furthermore, there is still another situation appears in the early development phase that is likely to be neglected. In this situation, sample data may be of incomplete information, for instance, we only have the information of WMC is 20, RFC is 35, Reliability is High, Maintainability is High, Performance is Low, but with no information about DIT. In order to use this incomplete information in the preceding model, it must be complemented with the information of DIT first. The complete information after compensation is WMC is 20, RFC is 35, DIT is Deep, Reliability is High, Maintainability is High, Performance is Low and WMC is 20, RFC is 35, DIT is Shallow, Reliability is High, Maintainability is High, Performance is Low. It should be emphasized that the sample data of DIT input from layer 2, while other variables’ information inputs from layer 1 and they meet together at the entrance of layer 3.

3 Conclusions

By integrating fuzzy logic into artificial neural network, the proposed synthetic prediction model can make quite accurate and effective description of complex and non-linear relationships between OO software internal attributes and quality attributes easily. Due to the fact that artificial neural network is good at handling quantitative data, while fuzzy logic is good at handling qualitative data, the integration of them is fit for the concurrence of multiple forms of data. In addition, artificial neural network can conduct learning and self-adaptation with only little information. Therefore it can be used in the training with incomplete information at the early stages of software development. As the development process goes on, more information will be available, which can be used
in further training of the model and adjusting previously obtained results, so that the prediction could be more accurate.

References


Brief Introduction to Author(s)

YAO Lan (姚 兰) was born in Sichuan Province, China, in 1982. She received B.Eng. degree from University of Electronic Science and Technology of China (UESTC) in 2004. She is now a postgraduate student with UESTC. Her research interests include signal and information processing.

YANG Bo (杨 拂) was born in Sichuan Province, China, in 1973. He received B.Eng. and M.Eng. degrees both from Xi’an Jiaotong University in 1995 and 1998, respectively. He received Ph.D. degree from National University of Singapore in 2002. From March 2001 to Sept. 2002, he worked full-time in Singapore Technologies Aerospace Ltd., engaged in software development process improvement and CMM certification of the company. From Sept. 2002 to Oct. 2003, he worked as a postdoctoral research fellow with Software Engineering Lab., School of Computing, National University of Singapore. He is currently an associate professor with School of Mechatronics Engineering, UESTC. His research interests include software reliability and industrial engineering.

(Continued from page 22)