Using neural network models in the quality management system for the software defect prediction

A D Danilov, D A Samotsvet and V M Mugatina
Voronezh State Technical University, Moscow ave., 14, Voronezh, 394026, Russia

E-mail: danilov-ad@yandex.ru

Abstract. Reasonable distribution of resources for regression testing execution of software is considered to be the most important task. Finding the best solution for it may significantly reduce expenses on the whole system development. Neural network model may be used for testing management, as it has fault-prediction ability in each program module. Code parameters are independent variables and presence of errors is a dependent value in such model. Neural network can learn on real data – real testing product. Testing results received from different environment may be integrated easily in the knowledge base. This allows neural network to learn during each testing iteration. The module that potentially contains an error is tested at the first place and more thoroughly. Presented method may predict testing results and distribute resources accordingly.

1. Introduction
Testing of software is resource and labor intensive process, time and staff demanding, which is at the same time an obligatory phase of software development lifecycle. That is why as many quality assurance processes as possible are desired to be automated.

Automated software testing is introduced on different project stages: code unit-tests automation for checking separate modules and their communication, automated server and databases requests, which are necessary for the product maintenance work. Load tests are designed for defining working capacity of the software in case of system usage by large number of users. Iteration and system tests are automated on user interface level.

Integration and system testing of user interface, especially in complicated and multifunctional software products is really time-consuming, but can not be skipped as it tests the system that further will be shown to customers and end users.

Already released and working functionality should not be influenced by the new developments implementation and should efficiently work after the program code is modified. Thus should be tested each time the team of developers updates the code.

This type of testing is called regression testing and is usually automated. Testing automation means in this case special scenarios development for each module in accordance with which particular course of actions for web and desktop application communication is executed and the particular set of checks is done in automatic mode.

For reasonable distribution of resources needed for regression testing neural network approach can be used, which allows to build the model of software behavior depending on implementation changes of the code and forecast errors in each specific module.
Received results of neural network testing allows prioritizing regression tests. This algorithm description can be found in section 2. To ensure neural network regression testing efficiency mechanism for knowledge base update should be developed. Test management system and database integration can be used as such mechanism, and thus knowledge base will be updated automatically with previous cycles tests results. Detailed description of this mechanism can be found in section 3.

2. Software fault-prediction using Neural Network

Running regression tests process can be very time consuming and as the new software version release connected with functions expansion or problems from previous versions solution may be urgent, regression testing of a very high quality should be done in a very short period of time. This is why the tests should be prioritized and put in a proper order for testing.

To solve these problems neural network system may be used, e.g. for building software product model, which can be used for forecasting potentially containing errors modules – code which has some inaccuracy in it, that may provoke system behavior different from expecting [1, 2]. And if results of that forecast show that error probability is higher than limit, such module should be tested at the first place.

All these shows that neural network-based model of software product has several independent variables and one dependent variable [3]. Dependent variable is module affiliation to the group of potentially faulty or on the contrary faultless modules. Numerous code characteristics can be used as independent variables, some of them may be found in table 1 [4, 5].

| Metric | Description                  |
|--------|------------------------------|
| LC     | Lines of code                |
| CC     | Cyclomatic complexity        |
| EC     | Essential complexity         |
| DC     | Design complexity            |
| BC     | Branch count                 |
| AC     | Afferent couplings           |
| EC     | Efferent couplings           |
| NPM    | Number of public methods     |
| AMC    | Average Method Complexity    |
| MA     | Measure of Aggregation       |
| MFA    | Measure of Functional Abstraction |

Neural network can learn on the real data, received after software testing. Each new iteration is initiated after new version of software preparation. Training set includes the values of used code metrics and it’s result – whether any errors/bugs in tested code were detected.

Originally the set of weights is defined randomly and then during the training the weights are set in way to provide a trustworthy behavior model of the software product, dependent from code changing, which provides back propagation algorithm [6]. Overall flowchart of learning of neural network and building a product model can be found in figure 1.

Received after training of the neural network model allows to forecast the fault in module of the software. The modules with higher fault containing possibility are tested more carefully while potentially faultless modules may be tested using means of basic regression testing only, which can reduce expenses needed for testing and the whole system development.

The model also helps to define the order of running the tests, the modules with fault probability higher than a limit are tested the first. This strategy may be used when the testing phase is limited in time and the maximum number of faults with the highest priority should be found in minimum time.
3. Knowledge base updating

Quality assurance management system needs to be continuously updated with regression testing results. Using log files as data sets for neural network learning can not be effective for machine learning based systems. Moreover, analysis of faults and failures found during regression testing is not automated in this case.

Integration of the test management system with the data warehouse is option that is more suitable to update data sets for neural network learning. This approach provides advanced functionality for event handling and can combine results of regression in different environments. All test results are stored in the data warehouse and represent a single knowledge base for further testing cycles. A detailed scheme of system integration can be found in figure 2.

General knowledge base generation algorithm provides flexible configuration of the data update method in the test management system. Furthermore, this architectural solution allows processing logged information about errors found during software testing.

The proposed approach reduces the effort to set up the integration of logged information and the neural network knowledge base even for the different environments.

The whole process of updating the knowledge base after iteration of regression testing is performed according to the algorithm shown on the figure 3 [7,8].

In addition, it is important to pay attention to processing the results of software testing and what advantages it can bring. As a rule, the software development team uses issue tracking system to control software development and bug fix process. Such systems involve creating tasks for each functionality and for fixing every detected fault in the product. Tasks contain a complete description, a
person who is responsible for it and how much time was spent on the implementation or bug fixing. Using the data warehouse with the modules for complex event processing provide the ability to automate the process of creating a bug issue, work with tasks in the product control systems and send reports to third-party systems. Figure 4 shows the process of working with the issue tracking system.

Figure 2. Component integration architecture.

Figure 3. Algorithm for updating the knowledge base.
Figure 4. Algorithm for complex QA system events processing.

This functionality provides a great opportunity for the redistribution of human resources within IT companies. The system allows to automate the work of analysts and quality assurance engineers. In addition, automated task creation can occur in real time, there is no need to wait for the end of the software product testing cycle.

4. Conclusion
This paper emphasis on the software failure prediction system which is based on neural network with back propagation learning algorithm. The model allows to make a prediction about the presence of fault in the module under test and optimally allocate resources for software regression testing.

The paper considers the knowledge base updating algorithm for an automated test management system. The standard mechanism has been improved by integrating the system with the data warehouse. The proposed approach provides an update of the data according to the results of previous test cycles. This solution allows you to analyze and process various system events in real time and optimize the performance of the system.

References
[1] Erturk E and Sezer A E 2015 Expert Systems with Applications 42(4) 1872-9
[2] Kanmani S, Uthariaraj R V, Sankaranarayanan V and Thambiduraia P 2007 Information and Software Technology 49(5) 483-92
[3] Pandey S K, Mishra R B and Triphathi A K 2018 Procedia Computer Science 132 1412-21
[4] Rajesh G, Kumar S, Kumar K, Mishra A and Catal C 2018 Computers & Electrical Engineering 67 15-24
[5] Radjenovic D, Hericko M, Torkar R and Zivkovic A 2013 Information and Software Technology 55(8) 1397-418
[6] Kleppmann M 2017 Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems (O'Reilly Media)
[7] Corr L 2014 Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema (DecisionOne Press)
[8] Cormen T H 2018 Introduction to Algorithms (Williams)