Prediction of Number of Faults And Time To Remove Errors

[1,]Nirvikar Katiyar [2,] Dr. Raghuraj Singh
[1,]*Ph.D. SCHOLAR J.J.T.U. RAJASTHAN*
[2,]*Prof. C.S. Department HBTI KANPUR*

Abstract:

*Advance knowledge of which files in the next release of a large software system are most likely to contain the largest numbers of faults can be a very valuable asset. To accomplish this, a negative binomial regression model has been developed and used to predict the expected number of faults in each file of the next release of a system. The predictions are based on the code of the file in the current release, and fault and modification history of the file from previous releases. The model has been applied two large industrial systems, one with a history of 17 consecutive quarterly releases over 4 years, and the other with nine releases over 2 years. The predictions were quite accurate: for each release of the two systems, the 20 percent of the files with the highest predicted number of faults contained between 71 percent and 92 percent of the faults that were actually detected, with the overall average being 83 percent. The same model was also used to predict which files of the first system were likely to have the highest fault densities (faults per KLOC). In this case, the 20 percent of the files with the highest predicted fault densities contained an average of 62 percent of the system's detected faults. However, the identified files contained a much smaller percentage of the code mass than the files selected to maximize the numbers of faults. The model was also used to make predictions from a much smaller input set that only contained fault data from integration testing and later. The prediction was again very accurate, identifying files that contained from 71 percent to 93 percent of the faults, with the average being 84 percent. Finally, a highly simplified version of the predictor selected files containing, on average, 73 percent and 74 percent of the faults for the two systems. Defect tracking using computational intelligence methods is used to predict software readiness in this study. By comparing predicted number of faults and number of faults discovered in testing, software managers can decide whether the software are ready to be released or not. In this paper the predictive models can predict: **(i)** the total number of faults (defects), **(ii)** the number of code lines changes required to correct a fault and **(iii)** the amount of time calculated (in minutes) to make the changes in respective object classes using software metrics as independent variables. The use of neural network model with a genetic training strategy is introduced to improve prediction results for estimating software readiness in this study. Our prediction model is divided into three parts: **(1)** prediction model for Presentation Logic Tier software components **(2)** prediction model for Business Tier software components and **(3)** prediction model for Data Access Tier software components. Existing object-oriented metrics and complexity software metrics are used in the Business Tier neural network based prediction model. New sets of metrics have been defined for the Presentation Logic Tier and Data Access Logic Tier. These metrics are validated using two sets of real world application data, one set was collected from a warehouse management system and another set was collected from a corporate information system.*

Keywords **:**Software Readiness, Predictive model, Defect Tracking, N-tier Application, Prediction of number of faults, software fault analysis and data accumulation.

I. INTRODUCTION:

Reliability is a significant factor in quantitatively characterizing quality and determining when to stop testing and release software on the basis of predetermined reliability objectives. Our works belong to a class of software reliability models that estimate software readiness through gauging on the amount of unresolved

residual defects. In highly competitive commercial software market, software companies feel compelled to release the completely ready software. Their task is treacherous, treading the line between releasing poor quality software early and high quality software late [26]. If software is released too early, then customers will be sent poor quality code; if it takes too long before releasing software, it can avoid the problems of low quality, but run the risk of exceeding deadlines and being penalized for the late delivery. Therefore, the ability to predict software readiness is essential to any software for optimizing development resources usage and project planning. A variety of traditional statistical techniques are used in software reliability modeling. Models are often based on statistical relationships between measures of quality and measures of software metrics. However, relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches [31]. Artificial neural networks and genetic training strategy are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus are attractive for software quality modeling. The use of neural network model with a genetic training strategy is introduced to improve prediction results for estimating software readiness in this study. With many variables, it is quite difficult to find the exact combination of contributions of each of the variables that creates the best predictions [23]. Genetic algorithm provides the solution to overcome this difficulty [4]. It finds the best set of importance of input values on an arbitrary scale of 0 to 1. The importance of input values are a relative measure of how significant each of the inputs is in the predictive model. We can then decide to remove the variables with the lowest Relative importance of inputs values. Most software today is developed so as to be integrated into existing systems. New software components are replaced or added into existing system. They use N-tier application architecture which produces flexible and reusable application for distribution to any number of client interfaces. Since all tiers have clear separation, this allows the developers to plug each layer in and out without too much hassles and without limiting the technology used at each tier [24]. For example, re-definition of the storage strategy or changing of DBMS middleware will not cause disruption to the other tiers. Our predictive model is targeting at software that is designed and written using this type of architecture. Our research team has analyzed the fault reports of a Warehouse Application System (WAS) and an Information Management System (IMS) and classified the faults. These software fault reports are recorded by CAIB GmbH, Murrhardt Company and Hwafuh, Myanmar software development team respectively during system test and maintenance stages in recent years for the Warehouse Application System and the Hwafuh Information System. The classification scheme was based on the nature of these faults and their relevance to the application architecture. For example, the faults coming from the Business Tier are strongly related to object oriented features and are introduced by features such as inheritance and polymorphism. In the Data Access Tier, the faults are strongly related to the code are used to interface with the database. Therefore, our prediction model is divided into three parts: (1) prediction model for Presentation Logic Tier (2) prediction model for Business Tier and (3) prediction model for Data Access Tier. Although N-tire application architectures are widely used, currently none of the existing research studies on the prediction of number of software development faults is based on architecture layers. Our model was built on an application tier basis using neural network model with a genetic training strategy and the trained system is used to predict the number of software development faults to help the software managers in deciding whether their software is ready to be released or not.

II. RELATED WORKS:

One of the methods to predict software readiness is defect tracking. McConnell describe four methods in [26]. Residual defects are one of the most important factors that allow one to decide if a piece of software is ready to be released. In theory, one can find all the defects and count them; however it is impossible to find all the defects within a reasonably short amount of time. One possible technique that a software manager can use is to apply the software reliability models and thus estimate the total number of defects present at the beginning of testing [35]. According to [1], there are essentially two types of software reliability models: those attempts to predict: (i) from design parameters and (ii) from test data. Both types of models aim to predict residual errors of software under development. The first type of models are usually called “defect density” models and use code characteristics in traditional software codes such as lines of code, nesting of loops, external references, input/outputs, cyclomatic complexity and so forth to estimate the number of defects in the software. Our model belongs to this category. However, instead of using conventional software parameters, our model focuses on objected-oriented code characteristics and N-tier application architecture. Nowadays, software houses widely adopted N-tier architecture approach in software development and have successfully implemented software systems using current technologies such as .NET and J2EE [24]. The second type of models is usually called software reliability growth models. These models attempt to statistically correlate defect detection data with known functions (such as an exponential function) using parametric methods. If the correlation is good, the known function can be used to predict future behavior. Both reliability models are used to estimate software readiness by gauging on the residual defects. Software development faults are predicted using various types of

software metrics in various studies [8, 11, 13, 17 and 21]. In our earlier studies, software development faults were predicted using Object-Oriented Design Metrics and SQL Metrics [32, 33]. The results from these studies showed that neural network models had better prediction accuracy than regression models. Our research set out to answer three questions: **(i)** how many faults are remaining in the programs, **(ii)** how many lines are required to be changed in order to correct these errors and **(iii)** how much time is required for the above activities. By comparing predicted number of faults and number of faults discovered in testing, software managers can decide whether the software are ready to be released or not. Most of the software quality models such as [18, 29, 6, 23 and 19] for object-oriented systems predict whether a software module contain faults by using traditional statistical models. However, only identifying faulty modules is not enough to estimate the software readiness.

III. SOFTWARE FAULT ANALYSIS:

Previous work in software fault modeling can be classified into prediction of number of faults remaining in the system and accounting for the number of faults found in the system. Most of the work on prediction is done in connection with software reliability studies in which one first estimates the number of remaining faults in a software system, then uses this estimate as a predictor of the number of faults in some given time interval. (See Musa, et al. [27] for an introduction to software reliability.) Classic models of software faults [29, 31] are discussed in a survey of the early work on measuring software quality by Mohanty [26] which has a section on estimation of fault content. There are also many recently proposed models [30, 32]. These models estimate the number of faults that are already in the software. Our work differs from these studies in that we assume new faults are continuously being added to the system as changes are made. Our research is similar to previous empirical studies of software faults, e.g. [12, 17, 30, 34], where the aim is to understand the nature and causes of faults. In these studies, explanatory variables are identified in order to account for the number of faults found. Our work extends this by attempting to understand how the process of software evolution could have an effect on the number of faults found over time. In several of the cited studies, no actual model is articulated. Our goal is to build fault models based on these explanatory variables that are reasonably accurate and interpretable. Below we list some of the factors, cited in previous work, which were thought to be predictors of number of faults. These factors can be classified into two groups: product related and process-related measures. Within each group, we will list the measurements in that group that we used to try to predict fault potential, and describe how successful these measurements were.

IV. SOFTWARE METRICS:

In this paper prediction model is divided into three parts, therefore software metrics are defined for different three levels as:

4.1 DATA ACCESS TIER:

We have analyzed the Data Access Tier source code of N-tier applications. SQL commands are mainly composed in the Data Access Tier classes to perform these database operations. It allows the creation of a set of useful and specific routines to be able to perform insert, select, delete and update actions on database tables. It does not usually contain business logic or presentation items. Therefore, the metrics for the Data Access Tier are different from other tiers. For source code of this tier, the number of database operations statements which are invoked from Data Access Tier classes and the error corresponding with these operations are more involved than other operations. For example, retrieving wrong database records, insertion fail error, cannot update error etc. In such cases developers need to check and modify the corresponding SQL statements to correct the error. To measure the size of Data Access Tier source code, we should measure the number of database operations statements instead of other LOC metrics for the Data Access Tier. The following Data Access Tier metrics are proposed and used in this study. Metrics having weak relationship with fault occurrence, such as the number of Data Definition Language (DDL) commands and Data Control Language (DCL) commands, are omitted in this study. Therefore we have measured three types of database operations, the statements given as (Table-1):

[1] The number of delete operations (TNDO) for deleting database records.
[2] The number of Select statements (TNSC) for data retrieval.
[3] The number of Insert/Update operations (TNIUO) for updating database records.

We have measured the complexity of SQL statements instead of other complexity metrics such as cyclomatic complexity because the Data Access Tier does not control the logic and flow of the application. These are controlled in Business Tier. The Data Access Tier codes only contain the specific routines to perform database operations. We have measured the complexity of SQL commands by using following construct that contain in SQL statements:

[1] The number of sub queries.
[2] The number of search condition criteria.
[3] The number of group by clauses.

Complexity of SQL statement is likely to increase following the increase in the number of these constructs because the latter can specify rows of data from a table or group of tables by joining such tables. As such, these are important in measuring the complexity of SQL statements as (Table-1).

Metrics	Description
TNDO	Total number of delete operations
TNSQ	Total number of sub-queries in data retrieval statements
TNIUO	Total number of insert/update operations
TNGB	Total number of group-by-clause in data retrieval statements
TNSC	Total number of select-SQL commands
ANSC	Average number of search condition criteria of where-clause in data manipulation statements

Table-1. List of Data Access Tier metrics

4.2 BUSINESS TIER:

It controls the logic and flow of the application. This layer does not have codes to access the database or codes for the user interface. Business Tier is the brain of applications since it basically contains elements such as business rules, data manipulation, etc. As this layer mainly contains object-oriented codes, object-oriented metrics are used for this tier. There is great interest in the use of the object-oriented approach to software engineering these days. Many measures have been proposed and evaluated in the literature to capture the structural quality of object-oriented code and design [6, 18, 19, 23, and 29]. The following existing traditional complexity metrics and object-oriented metrics are used in the prediction model for the Business Tier as (Table-2).

Metrics	Description
CBO	Coupling between objects
NOP	Number of parents
NA	Number of Attributes
AMC	Average Method Complexity
NOC	Number of Children
RFC	Response for a class
WMC	Weighted Methods per Class
LCOM	Lack of cohesion in methods
NMA	Number of methods added
DIT	Depth of inheritance tree

Table-2. List of metrics for the Business Tier

4.3 PRESENTATION LOGIC TIER:

Presentation logic tier works with the results/output of the Business Tier and handle the transformation into something usable and readable by the end user. It consists of windows forms, dialogs and ASP documents etc. We have found that some errors in this tier are strongly related to the interface objects. For example, improper display of edit box value, showing some buttons which should be hidden until some specific events have occurred, wrong adjustment of width of objects for some output values etc. In such cases developers need to check and modify the corresponding controls of these interface objects to correct the error. Therefore the number of user interface objects (NUIO) should be measured for predicting these kinds of errors instead of using other metrics such as LOC. Another metrics that has used in this tier is the total number of messages (TNM). This metric measures the total number of interface objects' messages of presentation layer classes. For example, BN_CLICKED message, LBN_DBCLICKED message of C list Box and C Button interface objects. When the user clicks a button, BN_CLICKED method is invoked. If an error occurred, the developers need to check the corresponding method to correct the error. A larger number of these methods are likely to correlate to

a higher number of fault occurrences. We use the following metrics for this tier of object-oriented applications instead of using metrics such as count of bytes, lines, language keywords, comment bytes, semicolons, block length, and nesting depth etc. (Halstead's metrics) – which are designed mainly for conventional applications with procedural flows.

4.4 Number of user interface objects (NUIO):

The NUIO is the measure of user interface objects or controls which handle input operations from user and output operation to users. For example, text box objects, button objects and label objects etc. The NUIO is defined as:

NUIO (c) = | Interface Objects (c) |

Where Interface Objects (c) is the set of interface objects which are declared in the Presentation Logic Tier class c.

4.5 Total Number of Messages (TNM):

This metric measures the total number of interface objects' messages of presentation layer classes. The TNM is defined as:

TNM (c) = | Messages (c) |

Where Message (c) is the set of messages which are handled by interface objects of the Presentation Logic Tier class c.

V. DEXTEROUS STUDY:

The genetic training strategy of Neuro Shell Predictor is used in this study. The genetic net combines a genetic algorithm with a statistical estimator to produce a model which also shows the usefulness of inputs. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data. The genetic method produces a set of relative importance factors that is more reliable than those produces by the neural method. Genetic algorithms (GAs) seek to solve optimization problems using the methods of evolution, specifically survival of the fittest. The functioning of the genetic estimator is built upon the General Regression Neural Net (GRNN) [5]. Genetic learning stores every set of inputs and related output in the training data. When predicting an output (e.g. defects) of particular input pattern, it is compared to every other pattern. Depending upon how close the match is, the output for each training row is weighted. Closer matches receive higher weights, and inputs that are farther away from the training inputs receive lower weights. The predicted output for the particular set of inputs is a "weighted" average of all of the other outputs (e.g. defects) with which the network was trained. As the genetic method uses a "one hold out" strategy during both training using current data set and afterwards when evaluating new data set, all available data sets can be used [25]. As such, out of sample evaluation set is not necessary when using genetic method. If the method is called upon to produce an output from a particular pattern of inputs T, then it never looks at T if T is in the training set, as long as "enhanced generalization" feature is turned on. This is true at all times during training. The genetic method is much like a "nearest neighbor" predictor or classifier. Its output is a kind of weighted average of the closest neighbors' outputs in the training set. In other words, if evaluating T, T is never considered to be in the neighborhood. Most other neural networks do not work this way. They look at the entire training set when being trained for each data set in the training set. Therefore, they are essentially looking at the entire training set when being applied to new data set later [23, 25].

5.1 Data Collection:

The experiment data is collected from two application systems. The first application is the warehouse management applications (WMA) that is developed using C, JAM (JYACC Application Manager) and PL/SQL languages. This set of applications has more than a thousand source files of C, JAM (JYACC Application Manager) and PL/SQL codes and uses the Oracle database. The warehouse system has been customized and used by many companies. Data Access Tier faults were collected from the journal files that contain the documentation of all changes in source files such as status of module, start date, end date, developer, nature of changes, etc. Data on software metrics were extracted from 102 PL/SQL files of the warehouse application. The second application used in this prediction is subsystems of an application system which is used in the Hwafuh (Myanmar) company, which is a fully networked information system. The Hwafuh Information System (HIS) contains more than 300 classes and approximately 1,000,000 lines of codes. The experiment data was collected from the payroll subsystem, time record subsystem, human resource function subsystem and piece calculation subsystems.

5.2 Experiments for the Data Access Tier:

Experiment Analysis 1:

The warehouse application system data was used for the experiment on prediction of data access faults of Data Access Tier. Six software metrics were extracted from 103 PL/SQL files. It contains TNSQ, TNSC, TNDO, ANSC, TNGB, and TNIUO metrics. The dependent variable was the number of Data Access Tier faults and the independent variables were the six software metrics identified above. First, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. A threshold value was set at 1000 for maximum number of generations without improvement. After 1621 generations, the optimized coefficient of multiple determination (R-square) value 0.737046 was arrived at. Therefore, about 74% of the variation in the number of faults can be accounted by the six predictors. To measure the goodness of fit of the model, coefficient of multiple determination (R-square), coefficients of correlation(r), mean square error (MSE) and root mean square error (RMSE) were used. The correlation of the predicted variable and the observed variable is represented by the coefficient of correlation (r). An r value of 0.861246 represents high correlations for cross-validation. The number of observations is 104. The significance level of a cross-validation is indicated by the p value. A commonly accepted p value is 0.06. In our experiment, a two tailed probability p value is less than 0.0001. This shows a high degree of confidence for the successful validations. The results clearly indicate a close relationship between Data Access Tier metrics (independent variables) and the number of Data Access Tier faults (dependent variable).

r (correlation coefficient)	0.860796
MSE	0.191002
t values	16.95783
R-square	0.768120
RMSE	0.4297
Avg error	0.267628
p values	< 0.0001

Table-3 Experimental result for the WMA system

5.3 Experiment Analysis 2:

The Hwafuh Information System (HIS) was used for prediction of data access faults, the number of line changed per class and required time (in minutes) to correct these data access tier faults. Six software metrics were extracted from 36 Data Access Tier classes.

5.4 Selection of Metrics for Data Access Tier:

Four Data Access Tier metrics (TNIUO, TNSC, ANSC and TNDO) from proposed six Data Access Tier metrics are selected in this experiment. TNGB and TNSQ metrics could not be collected because corresponding SQL constructs are seldom used in the HIS system.

5.5 Prediction of number of faults:

Average number of search condition criteria of where-clause (ANSC) and Number of insert /update operations (TNIUO) are found to be more important for number of faults prediction for the Data Access Tier. That shows the complexity of search condition criteria is highly related to occurrence of faults in this Tier.

5.6 Prediction of number of lines changed per class:

In this prediction, the TNSC metric is the most important inputs in this model. The amount of Select operation containing in Data Access Tier class is highly related to number of lines changes in that class. The ANSC and the TNIUO metrics are also important for prediction of number of lines changed.

5.7 Prediction of required time (in minutes) to change:

The TNIUO and the ANSC metrics are also most important for prediction of required time (in minutes) to change. Although only 36 rows of data have been tested in Experiment 2 the two-tailed P values are less than 0.0001. By conventional criteria, these p values are considered to be statistically significant. Experiment results show that r values are 0.788908 for prediction of the number of faults, 0.793297 for prediction of the number of lines changed per class and 0.818494 for prediction of required time (in minutes) to change. These results show a close relationship between Data Access Tier metrics (independent variables) and dependent variables (the number of faults per class and the maintenance cost). The values of squared multiple correlation are 0.732894 for prediction of the number of faults, 0.623987 for the prediction of the number of lines changed per class and

0.663987 for prediction required time (in minutes) to change. Therefore, about 72%, 63% and 67% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors.

5.8 Experiment for Business Tier:

The Hwafuh Information System (HIS) was used for prediction of data access faults, the number of line changed per class and required time (in minutes) to correct these Business Tier faults. Software metrics were extracted from 178 Business Tier classes.

5.9 Selection of Metrics:

As discussed in Section IV, we have selected the Chidamber and Kemerer object oriented metrics [29] and complexity metrics. After testing several combinations of models using Business Tier software metrics using subsystems of the Hwafuh Information system data, the following models achieved the best prediction for number of faults, number of line changes and required time (in minutes) respectively.

5.10 Prediction of number of faults:

The model used in this experiment contains NOP, RFC, DIT, LCOM, NMA, NOC, NA and CBO metrics. From Business Tier metrics, inheritance metrics (NOP, DIT), Response for a class (RFC) and cohesion metrics (LCOM) are the important ones for number of faults prediction in this tier.

5.11 Prediction of number of lines changed per class:

The model used in this experiment contains LCOM, DIT, NMA, RFC, NOC, WMC, CBO, NA, and NOP metrics. The cohesion metric (LCOM) is most important metric and other metrics is about equal importance.

5.12 Prediction model for required time (in minutes):

This model contains LCOM, DIT, NMA, RFC, NOC, WMC, CBO, NA, and NOP metrics. As in fault prediction experiment, LCOM, NOP, RFC and DIT are the most important. For 178 observations, the two-tailed p values are less than 0.0001. By conventional criteria, these p values are considered to be statistically significant. Experiment results show that r values are 0.838913 for prediction of the number of faults, 0.867894 for prediction of the number of lines changed per class and 0.748748 for prediction of required time (in minutes) to change the code. These results show a close relationship between Business Tier metrics (independent variables) and dependent variables (the number of faults per class and the maintenance cost / effort). The values of squared multiple correlation are 0.701936for prediction of the number of faults, 0.747582 for the prediction of the number of lines changed per class and 0.548946 for prediction of required time (in minutes) to change the erroneous code. Therefore about 70 %, 74% and 55% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors.

5.13Experiments for Presentation Logic Tier:

Presentation tier classes (58 classes) were collected from the Hwafuh Information System.

5.14 Selection of Metrics:

As they mainly include dialog classes, form classes and view classes etc, the number of user interface objects (NUIO) metrics and the total number of messages (TNM) metrics were proposed and used in this study. Their relative importance is roughly equal for experiments of Presentation Logic Tier.

5.15 Prediction of number of faults:

After performing 380 generations, an R square value of 0.67399 was received using the genetic learning strategy of Neuro Shell predictor. Therefore about 67% in the number of faults can be explained by the two selected predictors. An r value of 0.828206 represents high correlations for cross-validation for 58 observations.

5.16 PREDICTION OF NUMBER OF LINES CHANGED PER CLASS:

After performing 156 generations, an R square value of 0.53185 was received. Therefore about 52% in the number of lines changed per class can be explained by the two selected predictors. An r value of 0.740297 represents high correlations for cross-validation for 58 observations.

5.17 Prediction model for required time (in minutes):

After performing 116 generations, an R square value of 0.589987 was received. Therefore about 61% in required time (in minutes) to change per class can be explained by the two selected predictors. An r value of 0.800102 represents high correlations for cross-validation for 58 observations. From the experiment results, the Presentation Logic Tier metrics in this study appear to be useful in predicting software readiness. About 68%, 52% and 61% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors. The relative importance both Presentation Logic Tier metrics are about the same. Equal care should be taken when designing interface objects and messages of Presentation Logic Tier classes.

VI. CONCLUSION FOR COMPARATIVE STUDY USING REGRESSION MODEL:

We run the whole set of experiment using a regression model once again after execution. The superiority of the neural network model with genetic training algorithm in predicting software readiness using metrics is undisputable as it consistently performed the regression model.

VII. RESULTS:

In this section we present various models of modules' fault potential. First we discuss the stable model, which predicts numbers of future faults using numbers of past faults. Next we list out most successful generalized linear models, which construct predictors in terms of variables measuring the static properties of the code or comprising simple summaries of the change history using different models as: stable model and weighted time damped model etc.

VIII. CONCLUSION:

Our research extended currently software quality prediction models by including structural/architecture considerations into software quality metrics. The genetic training strategy of Neuro Shell Predictor is used in our study. The Genetic Training Strategy uses a “genetic algorithm” or survival of the fittest technique to determine a weighting scheme for the inputs. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data. Data patterns from WMS (103 patterns) were used for the prediction of the number of faults in the Data Access Tier prediction model. This study used 36 Data Access Tier classes, 178 Business Tier classes and 58 for Presentation Logic Tier classes from HIS to predict the number of faults, the number of lines changed and time required (in minutes). For these numbers of patterns, the genetic training method is much better because the evaluation set is not necessary in the genetic training method. We have compared this approach to a multiple regression analysis approach. The comparative performance data for the models are define by comparing empirical results including R2 and RMSE, Genetic Nets give better prediction results than Regression analysis. Regression analysis was done by using SPSS software. We intend to extend this investigation to a wide range of applications and also to propose a new set of Presentation Logic Tier metrics related to various types of user interface objects. For example, by separating pure display interface objects from interface objects that are able to perform both input/output functions. Also, it may be useful to separate control interface objects from interface objects which can be control objects as well as holding values.

REFERENCES:

[1] Alen Wood, “Software Reliability Growth Models”, hp Technological Report, TR-96.1, September 1996.

[2] Bellini, P. (2005), “Comparing Fault-Proneness Estimation Models”, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), vol. 0, 2005, pp. 205-214.

[3] D. A. Christenson and S. T. Huang, Estimating the fault con-tent of software using the _x-on-_x model," Bell Labs Technical Journal, vol. 1, no. 1, pp. 130{137, Summer 1996.

[4] D. E Goldberg, “Genetic Algorithms in Search, Optimization, and Machine Learning”, Reading, Mass: Addison-Wesley, 1989.

[5] D.F, Specht, “A general regression neural network”, IEEE Transactions on Neural Networks, vol. 2, Issue: 6, pp. 568-576, 1991.

[6] El Emam, W. Melo, C.M. Javam, “The Prediction of Faulty Classes Using Object-Oriented Design Metrics”, Journal of Systems and Software, Elsevier Science, 2001, pp. 63-75.

[7] Fenton, N. E. and Neil, M. (1999), “A Critique of Software Defect Prediction Models”, Bellini, I. Bruno, P. Nesi, D. Rogai, University of Florence, IEEE Trans. Softw. Engineering, vol. 25, Issue no. 5, pp. 675- 689.

[8] Giovanni Denaro (2000), ”Estimating Software Fault-Proneness for Tuning Testing Activities” Proceedings of the 22nd International Conference on Software Engineering (ICSE2000), Limerick, Ireland, June 2000.

[9] G. J. Schick and R. W. Wolverton, An analysis of competing software reliability models," IEEE Trans. on Software Engineering, vol. SE-4, no. 2, pp. 104{120, March 1978.

[10] J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability, McGraw-Hill Publishing Co., 1990.

[11] J. Jelinski and P. B. Moranda, Software reliability research," in Probabilistic Models for Software, W. Freiberger, Ed., pp. 485{502. Academic Press, 1972.

[12] K. H. An, D. A. Gustafson, and A. C. Melton, A model for software maintenance," in Proceedings of the Conference on Software Maintenance, Austin, Texas. September 1987, pp. 57{ 62, IEEE Computer Society Press.

[13] Khoshgoftaar, T. M. and J. C. Munson, (1990). “Predicting Software Development Errors using Complexity Metrics”, IEEE Journal on Selected Areas in Communications, 8(2): 253 -261.

[14] Khoshgoftaar, T.M., K. Gao and R. M. Szabo (2001), "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction. Software Reliability Engineering", ISSRE 2001. Proceedings of 12th International Symposium on, 27-30 Nov. (2001), pp: 66 -73.

[15] L Briand, W.L Melo, J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects", IEEE Transactions on Software Engineering, vol. 28 pp. 706 –720, 2002.

[16] Lanubile F., Lonigro A., and Visaggio G. (1995) "Comparing Models for Identifying Fault-Prone Software Components", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, June 1995, pp. 12-19.

[17] L. Hatton, \Reexamining the fault density{component size con- nection," IEEE Software, pp. 89{97, March/April 1997.

[18] M. Cartwright and M. Shepperd, "An Empirical Investigation of Object-Oriented Software System", IEEE Transactions on Software Engineering, vol. 26, pp. 786-796, 2000.

[19] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen,"An empirical study on object-oriented metrics", Proceedings of the Sixth IEEE International Symposium on Software Metrics, pp. 242-249, 1999.

[20] Manasi Deodhar (2002), "Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems", MS Thesis, Michigan Tech. University, Dec. 2002.

[21] Menzies, T., K. Ammar, A. Nikora, and S. Stefano, (2003), "How Simple is Software Defect Prediction?", Journal of Empirical Software Engineering, October (2003).

[22] Munson, J. C. and T. M. Khoshgoftaar, (1992), "The detection of faultprone programs", IEEE Transactions on Software Engineering, 18(5): 423- 433.

[23] NeuroShell Predictor Reference Manual, Ward Systms Group, Inc. http://www.wardsystems.com.

[24] Robert Chartier, "Application Architecture: An N-Tier Approach ", http://www.15seconds.com.

[25] C.H. Chen, "Fuzzy Logic and Neural Network Handbook", New York, N.Y.: McGraw-Hill, Inc., 1996.

[26] S. McConnell, "Gauging software readiness with defect tracking", IEEE Software, Vol. 14, no. 3, pp. 136, 135,1997.

[27] S. G. Eick, C. R. Loader, M. D. Long, L. G. Votta, and S. Van- derWiel, Estimating software fault content before coding," in Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, May 1992, pp. 59{65.

[28] S. N. Mohanty, \Models and measurements for quality assessment of software," ACM Computing Surveys, vol. 11, no. 3, pp. 251{275, September 1979.

[29] S.R. Chidamber, and C.F, Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.

[30] T. J. Yu, V. Y. Shen, and H. E. Dunsmore, An analysis of several software defect models," IEEE Trans. on Software Engineering, vol. 14, no. 9, pp. 1261{1270, September 1988.

[31] T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, S. J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system", Neural Networks, IEEE Transactions on ,Vol. 8, Issue 4, July 1997, pp. 902 – 909.

[32] Tong-Seng Quah, Mie Mie Thet Thwin, "Prediction of Software Development Faults in PL/SQL Files Using Neural Network Models", Information and Software Technology, vol. 46, No.8, pp. 519-523, 2004.

[33] Tong-Seng Quah and Mie Mie Thet Thwin, "Application of neural networks for software quality prediction using object-oriented metrics", Proceedings of International Conference on Software Maintenance, ICSM 2003, 22-26 Sept., 2003, pp. 116 – 125.

[34] V. R. Basili and B. T. Perricone, Software errors and complex- ity: An empirical investigation," Communications of the ACM, vol. 27, no. 1, pp. 42{52, January 1984.

[35] Y.K. Malaiya and J. Denton, 'Estimating the number of Residual Defects', HASE '98, 3rd IEEE Int'l High-Assurance Systems Engineering Symposium, Maryland, USA, November 13-14, 1998.