M. Zhao, C. Wohlin, N. Ohlsson and M. Xie, "A Comparison between Software Design and Code Metrics for the Prediction of Software Fault Content", Information and Software Technology, Vol. 40, No. 14, pp. 801-809, 1998.

ELSEVIER

Information and Software Technology 40 (1998) 801–809

INFORMATION AND SOFTWARE TECHNOLOGY

A comparison between software design and code metrics for the prediction of software fault content

M. Zhao[a], C. Wohlin[b,*], N. Ohlsson[c], M. Xie[d]

[a]*Department of Technology, University College of Gävle-Sandviken, Gävle-Sandviken, Sweden*
[b]*Department of Communication Systems, Lund University, Lund, Sweden*
[c]*Department of Computer and Information Science, Linköping University, Linköping, Sweden*
[d]*Department of Industrial and Systems Engineering, National University of Singapore, Singapore, Singapore*

Abstract

Software metrics play an important role in measuring the quality of software. It is desirable to predict the quality of software as early as possible, and hence metrics have to be collected early as well. This raises a number of questions that has not been fully answered. In this paper we discuss, prediction of fault content and try to answer what type of metrics should be collected, to what extent design metrics can be used for prediction, and to what degree prediction accuracy can be improved if code metrics are included. Based on a data set collected from a real project, we found that both design and code metrics are correlated with the number of faults. When the metrics are used to build prediction models of the number of faults, the design metrics are as good as the code metrics, little improvement can be achieved if both design metrics and code metrics are used to model the relationship between the number of faults and the software metrics. The empirical results from this study indicate that the structural properties of the software influencing the fault content is established before the coding phase.

Keywords: Software metrics; Correlation analysis; Fault prediction; Metric selection; Regression analysis

1. Introduction

Faults and failures in software are costly factors [1, 2]. They account for a significant amount of any project budget as activities related to fault detection and fault correction often correspond to 30%–50% of the budget [3]. This cost cannot be removed completely as methods are needed to ensure the quality of the software, but the costs for fault handling should be possible to decrease considerably by introducing and improving methods for early fault identification. Moreover, this information can be used as feedback to the development and the enhancement of the development processes. Thus, it can be used for process improvement and hence cost reduction.

Based on the above reasoning, it is clear that methods are needed to predict, control and improve fault handling in general. The type of methods can be divided into two major classes: methods for prediction of the number of faults in a specific module and methods for identification of fault-prone modules. The first type of methods has been investigated and evaluated, but it has been difficult to develop a valid model, in particular a model which is transferable between projects or organisations. An example of this approach is presented in [4]. The second type of methods has been more successful [1], and in particular it has been noted that we have a Pareto principle, i.e. a limited number of modules account for a considerably amount of the faults. In [1], it is shown based on fault data collected that 20% of the modules account for 60% of the faults for a large software system and how it is possible to build a transferable model between releases,

Thus, methods for identification of fault- and failure-prone modules and models for fault prediction are a potential way to improve software quality and to reduce cost. It is well known that a great deal of the logic in a software system is decided in the design phase, hence it is not unreasonable to assume that a model may as well be formulated from the design as from the code. The main focus of our previous studies [1, 5] has been on formulating models from design metrics. Our hypothesis is that a model for identification of fault-prone modules using design metrics is equally good as a model created based on code metrics. This hypothesis has, however, not been thoroughly evaluated. The underlying assumption has been that design metrics are as good as code metrics from a prediction

* Corresponding author. Tel.: + 46 46 222 3329, fax: + 46 46 145823; e-mail: claesw@tts.1th.se

0950-5849/98/$ - see front matter
PII: S0950-5849(98)00098-6

Table 1
Design measures collected

SDL-pages	Number of design pages in terms of SDL diagrams.
Tasks	Number of task symbols in the SDL descriptions.
Outputs	Number of output symbols in the SDL descriptions.
Inputs	Number of input symbols in the SDL descriptions.
If	Number of if-boxes in the SDL descriptions.
States	Number of state symbols in the SDL descriptions.
McCabe	This measure is an adaptation of the original proposal by McCabe. The measure is adapted to enable measurement on design and in particular to include the signalling concept used in SDL.
External inputs	Number of unique external inputs to a module.
External outputs	Number of unique external outputs to a module.
Internal signals	Number of unique internal signals within a module.

Table 2
Code measures collected

LOC	Number of lines of code.
Var	Number of variables in the code.
Retrieve	Signal sending of a so-called combined signal (application dependent).
Enter	Signal reception in the code.
Send	A signal sending in the code.
If…goto	Conditional goto in order to break an if-construct (see above).
Branch on	Conditional jump based any defined condition in the program.
If	Number of if-statements in the code.
Goto	Unconditional goto (see above).

point of view, and superior in terms of when the prediction can be made.

The objective here is to study if the model based on design metrics is as good as the model based on code metrics. In this paper, we focus on investigating the goodness of models for fault prediction using design metrics, code metrics and a model with a combination of design and code metrics. The latter means using information from both design and code to formulate a fault prediction model. The models are formulated based on data collected from the development of a software system in the telecommunications domain.

The paper is organised as follows. In Section 2, the background and their correlation analysis of the collected data are described, and in Section 3 the different prediction models are presented. Their detailed statistical analysis is given in Appendix B (Tables B1–B3). The interpretation of the models and evaluation of the results are discussed in Section 4. Finally, the findings are summarised and discussed in Section 5.

2. Data description and correlation analysis

2.1. Data description

The study is based on an investigation of 28 software modules for a large real time system in the telecommunication domain. The system studied is one of the company's main products. The actual release investigated is rather typical for releases of this particular system, although the choice of system and release was done based on availability.

The system consists of approximately 100 modules. Of these modules, 28 had full documentation required for this study, and were therefore included in this study. Other models were typically not updated to the actual implementation and could therefore not be used. The size of the modules is between 270 and 1900 lines of code. For each of these modules the number of fault reports from test and operation were counted and several design and code measures were collected. These types of measures are frequently referred to as complexity measures, but primarily they describe different aspects of the software and no measure can really be considered to capture the actual complexity. In total 19 different measures were collected, with 10 measures being collected from the design documentation and nine measures (including the number of lines of code) being collected from the resulting code.

The design data collected were primarily a counting of the number of symbols of different types. The language used during design is a predecessor to SDL, (specification and description language) [6], and it is basically a finite-state-machine with states and signals. A modified version of McCabe's cyclomatic complexity, [7] was used as well. The modification simply enabled signal sending and reception, respectively. The design measures are listed in Table 1.

The measures from the code are also primarily a matter of counting the number of constructs of different types, for example, number of variables and number of if-statements. The programming language is company internal and tailored to their applications and hardware platform. It must be noted that the programming language was very much based on goto-statements, and in particular it was observed that two different types of goto-statements were used. The difference is:

- goto 11; (unconditional goto)
- if 'condition' then goto 12; (conditional goto)

These two types were calculated separately since the hypothesis was that the first type of goto contributed more to the fault content than the second type of goto. The code metrics are briefly described in Table 2.

2.2. Comparing design and code metrics

Most studies only explore the statistical relationship between metrics, while vital expert knowledge about the metrics is neglected. Such information is essential for interpretation of analysis and the resulting models. The logical relationship between code and design metrics are summarised in Table 3.

Table 3
Logical relationships between metrics

SDL-pages vs LOC	These are highly related as both are size measures.
SDL-pages and LOC vs all measures	All measures more less increase the size of the modules, since they are countable, and hence they are highly related to size.
Outputs vs External outputs	Outputs describes the number of symbols and External outputs is the declarations of unique signals.
Inputs vs External inputs	See Outputs vs External outputs.
Inputs vs Enter	The Enter statement is primarily the implementation of the Inputs. Some differences may occur but basically there is a clear correspondence between these concepts.
Outputs vs Send	See Inputs vs Enter.
If (in design) vs If (in code)	See Inputs vs Enter.

The information in Table 3 should act as a complement to the correlation study presented below. It should, however, be noted that there is of course clear relationships between some concepts from design with concepts in code. The situation may, however, occur that new concepts are introduced and they may be a combination of design concepts, hence there may be situations which may be difficult to explain logically, and in these situations the correlation studies are invaluable.

2.3. Correlation analysis

Software metrics are often strongly interrelated or demonstrate a high degree of multicollinearity [8]. The correlation analysis for our data is presented in Table 4, and some of them are highly intercorrelated. However, some of these high correlations are due to the same type of metrics selected in two different phases, for example, Inputs and Enter, and should therefore have more or less the same value. The intercorrelations imply a need for data screening and reduction techniques. Factor analysis and especially principal components have gained some interest in the software engineering literature to handle the intercorrelation. However, there exists a number of problems with such techniques [9]. First of all, it is necessary that the principal components underlying the software complexity data are stable across software products, which appear not to be the case [10]. Principal components also require that the

Table 4
Correlations between the number of faults and metrics and intercorrelations among metrics

	Faults	SDL pages	Tasks	Outputs	Inputs	If	States	McCabe	External inputs	External outputs
Faults	1.000									
SDL pages	0.520	1.000								
Tasks	0.419	0.811	1.000							
Outputs	0.524	0.836	0.445	1.000						
Inputs	0.601	0.856	0.573	0.948	1.000					
If	0.549	0.893	0.891	0.649	0.728	1.000				
States	0.393	0.642	0.366	0.702	0.690	0.485	1.000			
McCabe	0.542	0.926	0.912	0.737	0.849	0.914	0.591	1.000		
External inputs	0.620	0.722	0.338	0.918	0.911	0.535	0.820	0.655	1.000	
External output	0.601	0.787	0.411	0.949	0.936	0.619	0.754	0.709	0.981	1.000
Internal signals	0.602	0.493	0.386	0.414	0.464	0.426	0.687	0.483	0.519	0.426
LOC	0.614	0.878	0.687	0.722	0.761	0.785	0.534	0.794	0.690	0.753
Var	0.674	0.837	0.679	0.752	0.787	0.716	0.589	0.807	0.709	0.725
Retrieve	0.155	0.523	0.538	0.329	0.267	0.444	0.316	0.477	0.218	0.282
Enter	0.645	0.694	0.302	0.897	0.890	0.527	0.818	0.622	0.989	0.961
Send	0.543	0.819	0.417	0.926	0.860	0.660	0.673	0.662	0.890	0.936
If…Goto	0.571	0.781	0.687	0.614	0.727	0.764	0.420	0.781	0.610	0.681
Branch on	0.598	0.709	0.568	0.675	0.787	0.589	0.716	0.754	0.790	0.760
if	0.534	0.788	0.698	0.625	0.750	0.766	0.479	0.806	0.626	0.688
Goto	0.662	0.721	0.586	0.603	0.719	0.678	0.431	0.714	0.644	0.688

	Internal signals	LOC	Var	Retrieve	Enter	Send	If…Goto	Branch on	If	Goto
Internal signals	1.000									
LOC	0.468	1.000								
Var	0.630	0.882	1.000							
Retrieve	0.183	0.556	0.557	1.000						
Enter	0.557	0.664	0.675	0.132	1.000					
Send	0.391	0.778	0.697	0.380	0.871	1.000				
If…Goto	0.391	0.937	0.774	0.400	0.595	0.645	1.000			
Branch on	0.622	0.760	0.808	0.315	0.776	0.608	0.752	1.000		
If	0.379	0.921	0.777	0.415	0.610	0.631	0.986	0.805	1.000	
Goto	00.405	0.914	0.803	0.329	0.638	0.619	0.965	0.810	0.955	1.000

data are scaled, while the models cannot be used to explain variations outside the data range from the given data set. Further, we strongly believe that a prerequisite for accepting any model is that we can interpret, understand and explain why a model is good, which is very difficult with models based on principal components. The correlation analysis in Table 4 is therefore essential when the prediction models are interpreted. Even though there are demarcations with regression models, such as unstable correlation coefficients, we believe such models are more appropriate for our objectives.

Table 4 was made from 27 modules because when all modules were used, the correlations between the number of faults and the metrics are very low (only two figures larger than 0.5, but still less than 0.6). Note that the main objective of this paper is to build prediction models and in particular to compare different models derived based on design metrics, code metrics and a combination of design and code metrics, respectively. Thus, we have taken the decision to remove one outlier, i.e. one module was deleted in order to provide a better basis for comparison. We would like to capture the general behaviour and goodness of models derived using different metrics, hence we would like to avoid that the models are governed by one or two individual data points. This is our motivation for removing one outlier.

When the correlations provided in Table 4 were studied, we noted that the metrics generally do not have very high correlations with the number of faults. For example, the correlation coefficient between *Retrieve* and the number of faults is 0.155. We believe that metrics such as *Retrieve* are not useful in building a linear regression model of the fault content. Therefore, a threshold of 0.5 was selected for screening the metrics before a formal analysis took place. Specifically, three metrics, *Tasks*, *States*, *Retrieve*, were deleted from building the prediction model according to this threshold.

3. Model building

In this section, we consider selecting the best regression equations using the design metrics, code metrics and all metrics, respectively. The regression equations are best in the sense that specific statistical procedures are applied for selecting variables in the regression. Note that there is no unique statistical procedure doing this. We use the stepwise selection procedure to build the regression models, see e.g. [11] for a detailed description of this procedure. It must also be noted that the main objective is relative comparison between the different models; the actual models are of minor interest in this paper.

3.1. Stepwise selection procedure

Let Y be the number of faults and X_1, X_2,…,X_k be the metrics concerned. We want to find, by the stepwise selection procedure, a linear model between Y and a subset of X_is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_q X_{q,q} \leq k. \quad (3.1)$$

The definitions and explanations of the statistical methods are provided in Appendix A (Table A1).

The stepwise selection procedure starts with selecting the metric with the highest correlation with Y and then the regression equation is determined. If an F-test shows that the regression equation is not significant, the procedure stops. Otherwise, this metric is retained and a search is conducted to identify the next metric to include in the model. At this stage, the partial correlation coefficients of the metrics not in the regression equation are examined. The metric with the highest partial correlation coefficient with Y is selected and a new regression equation is fitted. The regression is checked for significance, and the partial F-values for the variables in the equation are examined. The lowest partial F-value is compared with the predetermined F percentage point, and the corresponding metric is retained or rejected according to the test result. The procedure continues to search new predictors until no new metrics can enter the regression model.

It should be noted that at each stage, the partial F criterion for each variable in the regression should be checked. Therefore, one variable already in the regression may also be removed in the stepwise procedure.

3.2. Model using design metrics

When we first applied the stepwise procedure to the design metrics, an extremely large residual was found, see Fig. 1. This special observation also causes the same problem with the code metrics. Based on the same reason when the first outlier was deleted, we deleted this outlier from the following analysis.

For a given significance level 0.1, we carried out the stepwise procedure for the design metrics and two predictor metrics are selected from the eight design metrics (two removed due to low correlations with the number of faults). They are *External inputs* and *Internal signals*. The values of the parameters in the regression equation is given in Table 5.

For this regression model, the measure of goodness-of-fit (R^2-statistic) is 0.63. At the significance level 0.1, the critical value of F-test is 2.55. The F-value is equal to 19.5 that is much larger than the critical value of the F-test, so the regression is significant. More detailed results of statistical analysis are given in Appendix B.

3.3. Model using code metrics

Only one code metric is selected by the stepwise procedure, when using the same significance level of 0.1 as when building the design metric model. The model parameters are given in Table 6.

The F-value of 30.3 is much greater than the critical value of F-test which is equal to 2.93 in this case. The built model is therefore acceptable. The R^2-statistic is 0.558.

Table 5
Selected predictor design metrics and the coefficients

Design metrics	Const.	External input	Internal signal
Parameters	0.4978	0.0636	1.0726

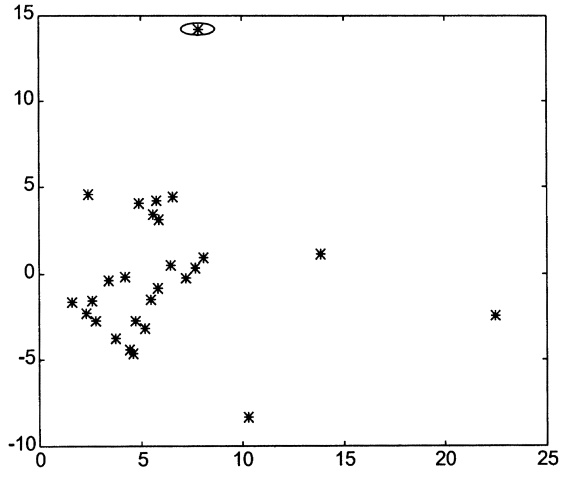

Fig. 1. Plot of residuals ($\hat{Y}_i$-Y_i) against the fitted values $\hat{Y}_i$ from the design metric model using 27 observations.

3.4. Model using both design and code metrics

Two metrics are selected from the 16 metrics, when the stepwise procedure with significance level of 0.1 is used. The model parameters are given in Table 7. In particular, we may note that the model includes one design metric and one code metric.

The *R*-statistic of this model is as high as 0.68 and the *F*-value is 24.6. The *F*-test shows that the regression is significant (critical value is equal to 2.55).

This model is better than the two previous models, since it gives a smaller estimate of the variance (the estimate is equal to 9.19) and large value of the *R*-statistic. In particular, the combined model is obviously better than the design model. Both the design model and the combined model have the same number of predictors, hence a direct comparison of the *R*-statistics and *F*-values is possible. We can, however, note that the improvement of the combined model is limited.

The comparison between the design model and code model is not straightforward because of different number of predictors. The design model has a higher *R*-statistic value and smaller variance estimate, but the code model has only one predictor. Therefore, it is reasonable, from a practical viewpoint, to conclude that the early estimate from a design model (at least for this particular data set) outweighs the other advantages with the code model.

Table 6
Selected predictor code metrics and the coefficients

Code metrics	Const.	Var
Parameters	− 1.9490	0.0576

Table 7
Selected predictor design and code metrics and the coefficients

Metrics	Const.	Internal signals	Var
Parameters	− 1.3907	0.8480	0.0352

4. Model interpretation

Model interpretation is not only essential, but also a prerequisite in any engineering discipline. If we are not able to understand and explain the outcome of quantitative analyses, the result and contributions of the work are both limited and questionable. The purpose of the model building is not only to identify cause-and-effects, but also to direct and support improvement efforts. Hence, few manager are prepared to take action based on a model that one cannot interpret.

The objective of this paper was to study the differences between design and code metrics, and their ability to predict the number of faults. Reviewing the correlations between the design metrics in Table 4, it is evident that no single metric can fully explain the variance, and that the metrics are highly intercorrelated. However, some of the metrics show a relative high correlation with faults, but a relative low correlation with each other, e.g. *External inputs* and *Internal signals*. If such metrics were combined they may be able to explain more of the overall variation. The result from the stepwise selection procedure for the design metrics includes these two metrics.

Interpretation design metrics: the values of the *R*-statistic in the models cannot directly be compared with the values in Table 4, but have some level of explanatory ability. That is, the combination of the two design metrics is better, but not drastically. Both metrics are based on extraction of signal handling, which in other studies have shown to be an area associated with severe problems [5]. However, our experience from root-cause-analysis and model building also indicates that the percentage of the faults associated with the logical modelling is high. It is therefore a bit surprising that metrics capturing the logical structure, e.g. *If* or *McCabe*, were not included.

Interpretation code metrics: the correlations in Table 4 show the same patterns for the code metrics as for the design metrics, however, the code metrics have slightly higher correlation with the number of faults. The only code metric included in the model is *Var*. Faults associated with variables and signals are logical faults — one of the most frequent classes of faults. Further, variables are only modelled during the coding phase. Therefore, it appears to be quite reasonable that *Var* was included in the model.

However, it was also a bit surprising that no metric, e.g. *Branch on*, capturing the logical structure was selected in this phase either. It has, however, a high correlation with *Var* which may explain why it is excluded from the model.

It is important to note that we do have high intercorrelations between variables, and we have chosen to include the variable with the highest correlation in the model although we might as well have chosen another variable as the two variables are highly correlated. Further, it is important to stress that prediction models of this type must be continuously evaluated and updated to take evolution into account. Moreover, we would like to emphasize that the objective is to compare design and code metrics, and not to find and recommend a particular model.

Comparing the separate design and code metrics analyses we conclude that design metrics are as good as code metrics in predicting the number of faults, even though the correlation values are not very high. However, given that only product metrics are included higher correlation values are not very likely to be found. To further improve the model, we believe that process and resource attributes should be included, rather than more refined metrics or extensive models based on product metrics.

Interpretation design and code metrics: in the final analysis (design and code model) two metrics were selected, one from each phase. The metrics capture different aspects and were identified in the previous analyses. Thus, it is quite reasonable that they have been included. This model is generally better than the previous two, it is hence preferable to use this model. However, the improvement compared with the design model is small, and the advantage of having a predictive model already in the design phase is obvious.

5. Summary and conclusion

We have conducted the comparison between design and code metrics based on data collected from a real project. By building a prediction model of the fault content, we can partially answer the questions discussed in the introduction. It should be noted that our conclusion is made based on one data set. It is hence difficult to claim that the results are general. To obtain general results, further studies are needed. It is reasonable to believe that similar results should be obtained for other releases of the same system, and also for other systems in the same environment being developed with the same process. The generalizability outside the company is, however, very difficult, but we believe that the findings are interesting enough for others to conduct similar studies. It is important to get estimations and predictions early, hence the findings indicate that such opportunities may exist.

Comparing the separate design and code metrics analyses, we conclude that design metrics are as good as code metrics in predicting the number of faults, although the correlation values are not impressing. However, little improvement can be obtained by considering the design metrics and code metrics together. This result indicates that a design metric model could be very useful since it can be applied much earlier than the code metric model. In particular, it indicates that the structural properties of the software influencing the fault content is mainly established already during the design phase. From this empirical study, we have seen that there is not much loss of information in terms of modelling the fault content if we ignore the code metrics.

To further improve the model we believe that process and resource attributes should be included, rather than more refined metrics or extensive models based on product metrics.

Appendix A. Concepts and symbols in data analysis

Table A1

Names and symbols	Description
$\hat{Y}_i$, $\hat{Y}_i$, $\hat{Y}$	Observations on responses, the fitted value and sample mean.
Sum of squares due to regression (SSD)	$\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2$ with the degree of freedom q − the number of parameters in regression model.
Sum of squares due to regression (SSR)	$\sum_{i=1}^{n} (\hat{Y}_i - Y)^2$ with the degree of freedom $(n - q - 1)$. The total variation of Y can be decomposed as the sum of SSD and SSR.
R^2	$\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2 / \sum_{i=1}^{n} (Y_i - \bar{Y})^2$ used as a measure for goodness of fit.
F-value, critical value, and F-test	$F = \frac{(SSD)/q}{(SSR)/(n-q-1)}, F(1-\alpha; q, n-q-1)$, the $100(1-\alpha)\%$ point of the F-distribution; When $F \geq F(1-\alpha;q,n-q-1)$, the regression is said to be significant under the significance level a.
Partial correlation between X_j and $X_1,\ldots,X_p$	Correlation between the residuals of Y regressed on $X_1,\ldots,X_p$, and the residuals of X_j, regressed on $X_1,\ldots,X_p$.
Partial F-value and test associated with a new variable X_j	$\frac{SSD\ including\ X_j - SSDwithoutX_j}{SSR\ including\ X_j)/\nu}$. When it is greater than $F(1-\alpha;1,\nu)$, the variable X_j is selected into the model. Here ν is the degree of freedom of the SSR including X_j.

Appendix B. Statistical analysis results

B.1. Design metrics model

The stepwise procedure produced the following model:

$$number\ of\ faults = 0.4978 + 0.0636 * External\ inputs + 1.0726 * Internal\ signals$$

The R^2-statistic, which is a measure for model goodness-of-fit, is 0.63. Table B1 displays the analysis of variance as usual.

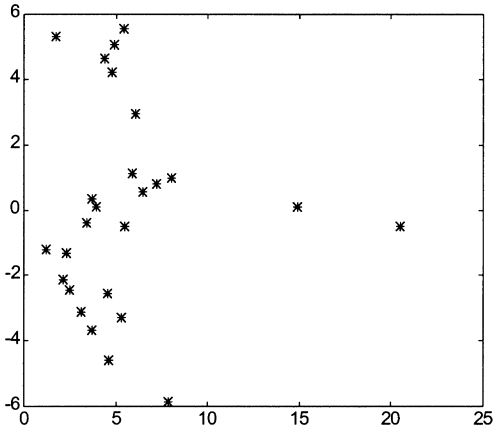

Fig. 2. Plot of residuals ($\hat{Y}_i$-Y_i) against the fitted values $\hat{Y}_i$ from the design metric model.

The estimate of the variance, made from the mean square of residuals, is 10.7. At the significance level 0.1, the critical value for F-test is 2.55. The F-test is 2.55. The F-value is equal to 19.5 which is much larger than the critical value, so the regression is significant.

If we study the plot of the residuals against the fitted values, see Fig. 2, there is a need to judge whether or not they are a sample from the normal distribution. The normal plot of the residuals displayed if Fig. 3 shows a good fit. Therefore, we can accept this model.

B.2. Model using code metrics

Using the stepwise procedure with significance level 0.1, only one code metric is selected. The model is

$$number\ of\ faults = -1.949 + 0.0576 \times Var$$

The F-value of 30.3 is much greater than the F-test critical value of 2.93. The model passes the overall F-test. The R^2 note statistic is 0.558.

The residual plot and normal plot indicate that the model is acceptable (Fig. 4 and Fig. 5).

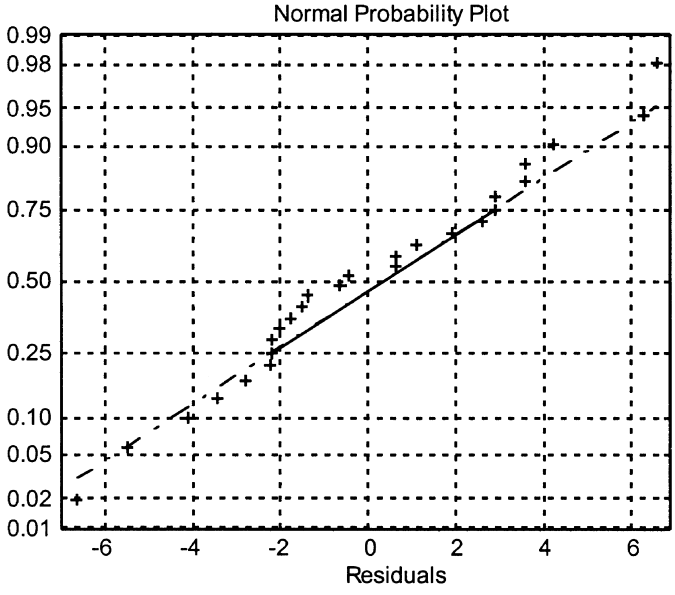

Fig. 3. Normal plot of residuals from the design metric model.

Table B1
Analysis of variance table for the design metric model

Source of variation	Degree of freedom	Sum of square	Mean square	F-value
Due to regression	2	416.4	208.2	19.5
About regression (residual)	23	246.1	10.7	
Total, corrected for mean $\bar{Y}$	25	662.5		

Table B2
Analysis of variance table for the code metric model

Source of variation	Degree of freedom	Sum of square	Mean square	*F*-value
Due to regression	1	369.8	369.8	30.3
About regression (residual)	24	292.6	12.2	
Total, corrected for mean $\bar{Y}$	25	662.5		

Table B3
Analysis of variance table for the design and code metric model

Source of variation	Degree of freedom	Sum of square	Mean square	*F*-value
Due to regression	2	451.2	225.6	24.6
About regression (residual)	23	211.3	9.19	
Total, corrected for mean $\bar{Y}$	25	662.5		

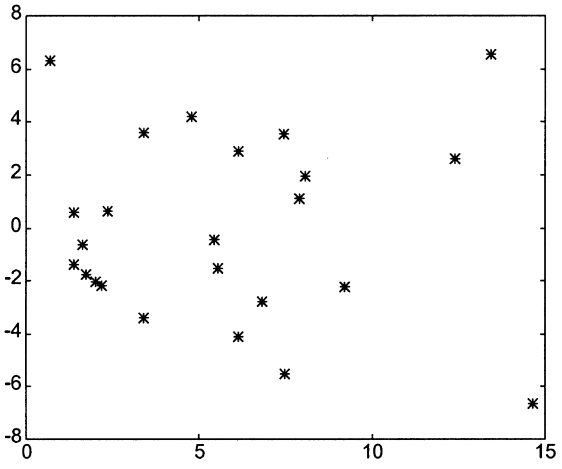

Fig. 4. Plot of residuals against $\hat{Y}_i$ from the code metric model.

B.3. Model using both design and code metrics

When the stepwise procedure with significance level of 0.1 is carried out for all metrics, two metrics are selected from the 16 metrics. In particular, we can note that the model includes one design metric and one code metric.

$$number\ of\ faults = -1.3907 + 0.848 \times internal\ signals + 0.0352 \times Var$$

The R^2-statistic is as high as 0.68 and the *F*-test shows that the regression is significant (*F*-test critical value 2.55 and *F*-value 24.6). The residual plot in Fig. 6 and normal plot in Fig. 7 indicate that this is an acceptable model.

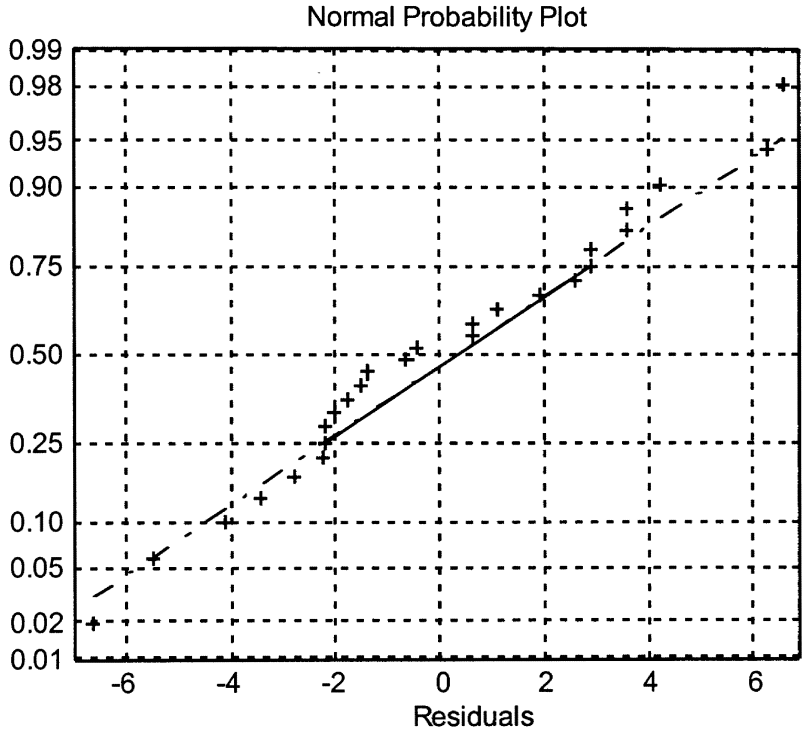

Fig. 5. Normal plot of residuals from the code metric model.

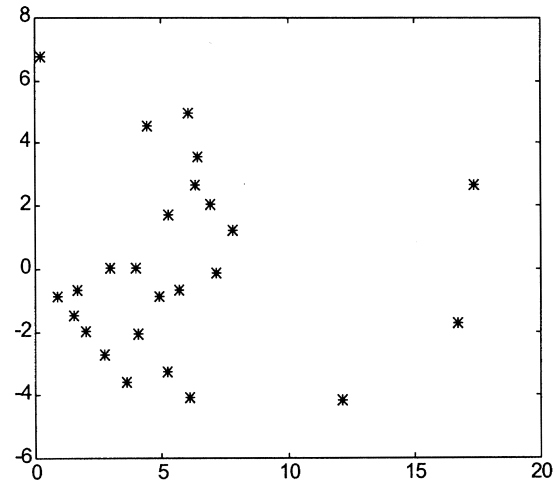

Fig. 6. Plot of residuals against $\hat{Y}_i$ from the design and code metric model.

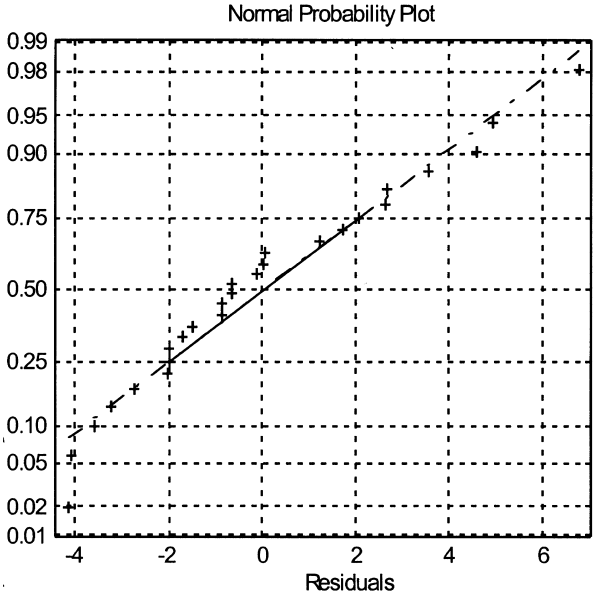

Fig. 7. Normal plot of residuals from the design and code metric model.

References

[1] N. Ohlsson, M. Helander, C. Wohlin, Quality improvement by identification of fault-prone modules using software design metrics. Proc. 6th International Conference on Software Quality, Ottawa, Canada, 1996, pp. 1–13.

[2] W. Gibbs, Software's chronic crisis. Scientific American, September (1994), 86–94.

[3] I. Sommerville, Software Engineering, Addison-Wesley, Reading, MA, 1996.

[4] Lennsellus, B. Software complexity and its impact on different software handling processes. Proc. 6th International Conference on Software Engineering for Telecommunication Systems, 1986, pp. 148–153.

[5] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE Transactions of Software Engineering 22 (12) (1996) 886–894.

[6] ITU, Recommendation Z100: SDL — Specification and Description Language.

[7] T.J. McCabe, A complexity measure, IEEE Transactions on Software Engineering 4 (2) (1976) 308–320.

[8] J.C. Munson, T.M. Khoshgoflaar, The detection of fault-prone programs, IEEE Transactions on Software Engineering 18 (5) (1992) 423–433.

[9] N. Ohlsson, M. Zhao, M. Helander, Application of multivariate analysis for software fault prediction. Technical Report LiTH-IDA-R-96-30, Linköping University, 1996.

[10] T.M. Khoshgoflaar, D.L. Lanning, Are the principal components of software complexity data stable across software products? Proc. of the Second International Software Metrics Symposium, October 24–26, London, England, 1994, pp. 61–72.

[11] N. Draper, H. Smith, Applied Regression Analysis. Wiley, New York, 1981.