Fault Content Estimations:
A Pragmatic Approach using Design Metrics

Claes Wohlin
Dept. of Communication Systems,
Lund University, Box 118,
S-221 00 Lund, Sweden,
Phone: +46-46-2223329,
Fax: +46-46-145823
E-mail: claesw@tts.lth.se

Min Xie
Industrial & Systems Engineering Dept.,
National University of Singapore,
10 Kent Ridge Crescent, Singapore 0511,
Phone: +65-7726536,
Fax: +65-7771434,
E-mail: isexiem@leonis.nus.sg

Abstract

The identification of fault-prone parts of a system or estimation of the total fault content are important for planning purposes and to increase the quality of the delivered software. This paper presents a pragmatic procedure for metrics selection to include in a prediction model of fault content. The proposed procedure is simple to apply and it is based on correlation analysis between the metrics, fault content and inter-correlation between the different metrics. Multicollinearity is addressed by the selection method, although not formally handled. A prediction model is derived from one data set and evaluated in a second data set. The model performs well, but more analysis is required. Future work also includes evaluating the pragmatic approach based on correlation with other approaches, such as stepwise regression analysis and principal components. The main objective of the study is to raise a number of questions for discussion and further work.

Keywords: Fault content, software metrics, complexity metrics, corrective maintenance

1. Introduction

Faults and failures in software 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 [Sommerville96]. This cost cannot be removed completely as methods are needed to at least ensure the quality of the software, but the costs for fault handling should at least be possible to decrease considerably by obtaining early estimates of the fault content that can be expected in a particular software system. An early estimate of the fault content would improve the predictability of test and maintenance effort.

Approaches for estimation of the number of faults in a particular module using complexity measures are difficult [Wohlin95]. The measures are unable to provide a complete explanation of the fault content, other factors such as test effort and dependence on the individual programmer who developed the module are not captured by the complexity measures. Other approaches suggested include identification of the most fault-prone modules [Ohlsson96], but this type of methods do not lend themselves to perform fault content estimations.

Identification of the most fault-prone modules are important, but individual estimates are difficult and we may end up with inaccurate estimations. The individual estimates may, however, be combined into an estimate of the total fault content of the system. A major advantage with one estimate based on the individual estimates is that under- and overestimates even out, and the hypothesis is that the combined estimate becomes acceptable as an indicator of effort needed for testing and corrective maintenance.

In this paper, we have used data from two systems. We have collected a number of design metrics from both systems. The design metrics and fault data from the first system are used to build a prediction model, and the model is then applied for the second system. The prediction model is derived using a pragmatic approach based on correlations between different measures. The model is evaluated from a goodness of fit point of view in the first system, and its predictive validity is evaluated in the second system.

The paper is organized as follows. Some background information concerning the data is given in Section 2. The derivation of the model is discussed in Section 3, including a quality of fit evaluation. The predictive model is then applied, which is discussed in Section 4. Finally, a discussion and some open questions are provided in Section 5.

2. Data collection

The study is based on an investigation of two large real time systems. For the first system we managed to retrieve design metrics for 28 software modules, and for the second system data were available for 28 modules too (by coincidence). The size of the modules varies between 270 to 1900 lines of code. For each of these modules the number of failure reports were counted and several design measures were collected. These type 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 10 measures were collected from the design documentation.

The design data collected are primarily counts of the number of symbols of different types. The language used during design is a predecessor to SDL, (Specification and Description Language) [ITU88], and it is basically a finite-state-machine with states and signals. A modified version of McCabe´s Cyclomatic Complexity [McCabe76] was used as well. The modification was simply due to being able to handle signals. The design measures are listed in Table 1.

TABLE 1. Design measures collected.

Measure	Explanation
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.

Furthermore, fault data for the modules were collected. The number of faults for a module ranges from 0 to 32. The first system is used to build a prediction model of the fault content and the model is then evaluated for the second system.

3. Building a prediction model

Prediction models of software fault content are normally built using a statistical model, for example a regression model, in conjunction with a selection method, for example, principal components [Khoshgoftaar94]. The methods are many times viewed by practitioners as rather difficult and hard to understand. Principal components are hard to grasp due to the grouping of metrics into some abstract classes which not necessarily have a direct mapping to design or code concepts, although there may be interpretations on a higher abstraction level. Another approach is to use linear regression or multiple linear regression directly. The linear regression technique is mostly too limited (one parameter is not enough as an explanatory variable), and for multiple linear regression we have to find which parameters to put into the regression formulae. Moreover, we would mostly like to avoid negative parameters. It is, for example, unreasonable to believe that a larger number of if-statements would decrease the number of faults. If a multiple linear regression model is created without constraints the best model may very well include negative parameters, which are difficult to explain. Thus, more sophisticated techniques are needed to create a multiple linear regression model [Zhao97], hence bringing us back to the problem of understandability for many practitioners.

Based on the above reasoning, we would like to investigate and propose a more pragmatic approach when selecting the metrics to include in a multiple linear regression model. The major advantage of the approach is that it is easy to understand and the method provides a practical explanation to why the parameters are included in the prediction model. The method is based on selecting metrics to include in the model using correlation coefficients after removal of outliers. Outliers are removed as they are interpreted as special cases for this particular data set. It is clear that special cases probably exist for the system for which we would like to perform predictions, but our proposal is that outliers require special care and must be handled separately from the “normal” cases. Outliers are always controversial when building prediction models and the issue requires further studies and analysis.

3 modules, out of the 28 modules, were removed as outliers, where an outlier is identified using a Box plot. The parameter used to identify an outlier is fault density, i.e. number of faults per lines of code. All three modules removed had more faults than expected, one module in particular was an extreme outlier.

The correlation between the design metrics and the number of faults for different modules, and the inter-correlations between different design metrics are shown in Table 2. Multicollinearity is a well-known problem when having highly correlated parameters from which we would like to derive a prediction model, beginning with selecting an appropriate subset of parameters to use in the prediction model. Multicollinearity is explicitly addressed by, for example, principal components and stepwise multiple linear regression. In our pragmatic approach, multicollinearity is addressed implicitly through the selection criteria. In particular, by step 2 below, where it is required that the next parameter to include should have higher correlation with the number of faults than its inter-correlation with the parameters already included in the model. The objective is to try to ensure that highly correlated measures are not included in the model, unless they have a higher correlation with the number of faults than with the parameters already in the model.

TABLE 2. Correlations between faults and metrics, as well as inter-correlation between metrics.

Parameter	**Faults**	**SDL-pages**	**Tasks**	**Outputs**	**Inputs**	**If**	**States**	**McCabe**	**Ext. Inputs**	**Ext. Outputs**
SDL-pages	0.602	1								
Tasks	0.441	0.819	1							
Outputs	0.574	0.834	0.449	1						
Inputs	0.639	0.857	0.574	0.947	1					
If	0.566	0.898	0.896	0.642	0.719	1				
States	0.540	0.645	0.381	0.715	0.711	0.504	1			
McCabe	0.594	0.928	0.913	0.735	0.850	0.915	0.603	1		
Ext. Inputs	0.638	0.724	0.331	0.919	0.908	0.517	0.855	0.651	1	
Ext. Outputs	0.604	0.794	0.405	0.954	0.935	0.604	0.713	0.708	0.988	1
Internal	0.776	0.484	0.403	0.409	0.465	0.433	0.681	0.490	0.531	0.438

The metrics to be included in the prediction model are derived using the following procedure:

1. The metric with the highest correlation with the number of faults is included in the model.
2. The next candidate metrics are those that have a higher correlation with the number of faults than their inter-correlation with the metrics previously included in the model. The candidate metrics are always chosen from previous candidates, i.e. if a metric is not a candidate in one evaluation then it is not included among the candidates in later evaluations. The metric with the highest correlation to the number of faults is included in the model.
3. Goto step 2 until no candidate metrics could be included in the model.

In this particular case, the result becomes as follows. First, “Internal” is chosen, due to the high correlation. The next candidates metrics are all metrics except “States”. This results in choosing “Inputs” (“External Inputs” may have been chosen as well based on the correlations in Table 2), which results in no further candidates to include. Thus, the multiple linear regression model should be derived using “Internal” and “Inputs”.

The major advantage with the above procedure for selecting metrics to include in the prediction model is the intuitive interpretation, i.e. to include metrics which have a higher correlation with the faults than with the metrics already included in the model. The procedure works in a similar way as a stepwise multiple linear regression method, but it makes the selection of metrics to include in the model more visible and understandable form a practical point of view.

The prediction model is derived from the data of the first system. The model becomes, after determining the model using multiple linear regression based on the selected measures, with R = 0.837 and is highly significant using an F-test ($p < 0.0001$):

$$Faults = -0{,}726 + 0{,}051 \times Inputs + 1{,}161 \times Internal$$

Using this formulae, the fault content of the first system is estimated to evaluate the quality of fit. The fault content is estimated to 148 faults, and the actual known fault content is 198 faults. This is a clear underestimation, and the reason is the outliers which were removed when deriving the model. Thus, it is no surprise that the model underestimates. The model is derived from “normal” modules and should be evaluated in a quality of fit of the data from which it was derived. The outliers have to be handled separately. If removing the outliers in the quality of fit analysis, then the estimate becomes 137 and the actual fault content is also 137. Thus, we must search for patterns among the outliers in order to perform a separate estimation for potential outliers when using the model for prediction. After a first brief inspection of the collected data there is no obvious reasons for why these three modules became outliers. Further analysis is required.

The objective in the future is to compare the pragmatic selection method with a stepwise regression approach and a multiple regression approach with principal components as the selection method.

4. Application of the prediction model (validation)

The model derived for the first system is now applied using the design measures for the second system. The estimated fault content becomes 255 fault, and the actual fault content is 234 faults. The estimate is surprisingly good, in particular based on the problems reported for the outliers of the first system. We are, of course, unable to remove outliers for this system as the information is not available at the point of prediction.

In retrospect, we are able to study the result if we were able to remove outliers at an early stage, for example, through identification of similar patterns between outliers for different systems. To gain further understanding regarding outliers; an outlier analysis was performed for the second system too. 2 outliers were identified, and if these are removed from the estimation then the estimate becomes 247, which should be compared with 195. The removal of outliers has been performed in retrospect, and so far we have not identified a good way of removing potential outliers based on module characteristics. In particular, we may note that the removal of outliers this time made the prediction worse. This may indicate that it is difficult to identify some common characteristics for the outliers.

It would be beneficial if we could:

- Build a model for “normal” modules, see Section 3.
- Identify characteristics of outliers and build separate models for outliers, or at least identify common patterns which are common across systems.

If we could succeed in this mission, we would probably be able to improve our ability to perform fault content estimations already from the design and hence providing a valuable tool for planning and controlling test effort and corrective maintenance.

5. Discussion

The pragmatic approach for selecting prediction model parameters seems to work rather well, and in particular it is easy to understand and apply for practitioners. Furthermore, the approach for predicting test effort and corrective maintenance seems worth investigating further.

The study leaves a number of open questions, for example:

- How good is the model derived using the pragmatic approach compared with models derived using other selection and prediction approaches?
- Can we identify some common characteristics for the outliers, hence providing a basis for separating “normal” modules from particular fault-prone modules? This would provide a better basis for estimation as discussed earlier.
- Can we by combining metrics (addition or multiplication or by other means) find a better model? How do we include other type of metrics, for example process metrics, in our parameter selection procedures and statistical predictions models?
- How advanced statistical methods should we apply on software engineering data? In particular, we must consider how more reliable data can be collected and how we present the statistical methods to software engineers in industry.

The quantitative study provides a starting point for estimating fault content to provide a basis for planning and controlling testing and corrective maintenance. It is, however, clear from this study that further analysis is necessary and that several issues have to be resolved to make the approach more accurate.

6. References

[Khoshgoftaar94] T. Khoshgoftaar and D. L. Lanning, “Are the Principal Components of Software Complexity Stable Across Software Products?”, Proceedings Second International Symposium on Software Metrics, pp. 61-72, London, UK, 1994.

[Ohlsson96] N. Ohlsson, M. Helander and C. Wohlin, “Quality Improvement by Identification of Fault-Prone Modules using Software Design Metrics”, Proceedings 6th International Conference on Software Quality, pp. 1-13, Ottawa, Canada, 1996.

[Sommerville96] I. Sommerville, “Software Engineering”, Addison-Wesley, 1996.

[Wohlin95] C. Wohlin, “Revisiting Measurements of Software Complexity”, Proceedings Asian Pacific Software Engineering Conference, pp. 35-43, Seoul, South Korea, 1996.

[Zhao97] M. Zhao, C. Wohlin, N. Ohlsson and M. Xie, “A Comparison between Software Design and Code Metrics for the Prediction of Software Fault Content”, Technical report, Linköping University, Sweden.