The Accuracy of Early Fault Prediction in Modified Code

Piotr Tomaszewski, Lars Lundberg, Håkan Grahn
Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
SE-372 25 Ronneby, Sweden
{piotr.tomaszewski, lars.lundberg,hakan.grahn} @bth.se

ABSTRACT

Software systems are normally developed in a number of releases. Each release usually modifies existing code. In this study we show that such modified code can be an important source of faults. Since faults are considered major cost drivers of software projects, the ability to identify fault-prone classes before they are implemented would give a chance to apply some preventive measures, which could bring significant savings on project costs. To achieve that, however, the prediction model available early in the development process would have to be accurate. In this study we compare the accuracy of fault prediction models available before and after the system is implemented. We find that fault prediction models that are available after the system is implemented are about 34% more accurate compared to models available before the system is implemented. We discover that the higher accuracy of the prediction models available after the system is implemented is caused by the metric that describes the size of the class modification. This metric is a code metric that is available only after the system is developed. As further work, we suggest defining design metrics that describe the characteristics of modifications and evaluating their applicability to predict faults in modified code.

Categories and Subject Descriptors

D.2.5 [**Testing and debugging**], D.2.8 [**Metrics**]

General Terms

Measurement, Verification.

Keywords

Fault prediction models, early fault prediction, modified code

1. INTRODUCTION

Faults are considered as one of the major cost drivers of software development. Finding and correcting them is a very expensive activity [20]. Fault detection can account for a significant part of the project budget, e.g., in the study reported in [2] 45% of the project resources were devoted to testing and simulation.

Typical software systems are produced in a number of releases. In each new release usually a significant amount of new functionality is added. This often results in major changes introduced to the code implementing the current system. Such code modifications are an important source of faults [16].

A well known fact concerning faults is that a majority of the faults can be found in a minority of the code (e.g., 60% of the faults can be found in 20% of the modules [14]). A prediction model that identifies the most fault-prone code can bring significant savings on project costs by enabling more efficient allocation of resources. This has been widely recognized by researchers that attempted to build prediction models, including models available early in development process (e.g., [5, 14, 16-18, 22]).

Fault prediction models are usually based on different characteristics of the software, e.g., design or code metrics (e.g., [5, 22]). Some of those metrics are available only after the system is developed. These are mostly code metrics, like the number of lines of code or McCabe complexity [8]. A model based on these metrics can be applied only after the system has been developed. Other metrics, e.g., design metrics, are often available before the coding has started, e.g., from the design documentation. Prediction models based on such metrics are able to identify fault-prone classes before they are implemented. This makes it possible to apply some preventive measures to such classes, like assigning them to more experienced developers or increasing the number of code reviews. This can lead to large potential savings. However, the savings largely depend on the accuracy of the model based on the metrics available early in the design phase.

In this paper we compare the accuracy of fault prediction models that are available before the system is developed with models that contain data available only after the system has been implemented. The models are built using linear regression and predict the number of faults in a class. We discover that models available early in the development process are about 34% more accurate compared to those available before the system is implemented.

Our models are based on the data from one release of a large telecommunication system developed by Ericsson. The system comprises about 600 classes (250 KLOC in total). The system is divided into subsystems. Each subsystem comprises a number of components. The system works in a service layer of a mobile phone network. It is mission-critical system for mobile network operators. Because of high quality requirements the system undergoes extensive, and therefore expensive, testing before it is released to the market. The system is a mature system that has been available on the market for over six years. The people involved in its development are experienced and have been involved in the development of the system for several years.

The release that we examine in this study is considered typical for this system. 40% of the code that is included in the examined release of the system is new compared to the previous release. 65% of the new code has been introduced as modifications of existing classes and 35% as new classes. The faults found in the modified code accounted for 86% of the total number of faults found in the new and the modified classes. Our goal was to build prediction models for the modified classes, as faults in those classes contribute most to the total number of faults in the release. The faults we discuss in this study are all the faults that had been reported until the project reached the stage of market availability.

The rest of the paper is structured as follows: in Section 2 we present the work that has been done by others in the area of fault prediction, Section 3 describes methods we have used for model building and evaluation. Section 4 presents the results. In Section 5 we discuss our findings. In the last section (Section 6) we present the most important conclusions from our study.

2. RELATED WORK

The most common method for building fault prediction models is multivariate linear regression (e.g., [2, 3, 13, 15, 18, 22]). Typically, the first step of model building involves some kind of selection of metrics for the model. The metrics used should be independent from each other. According to [2] introducing dependent metrics into the model causes a risk of multicolinearity. Examples of dependant metrics are number of lines-of-code and number of language statements, because they are very likely to measure the same thing – size - in different units. Multicolinearity is especially risky when regression models are built. It leads to "*unstable coefficients, misleading statistical tests, and unexpected coefficient signs*" [7]. Obviously, selected metrics should also be good fault predicators. Introducing metrics that are not good fault predicators into a regression equation increases the risk for misleading and unstable models.

The next steps involve model equation calculation and evaluation. A typical evaluation criterion for regression models is "goodness-of-fit" measure called R^2. R^2 describes the proportion of variability of variable predicted by the model [11]. It has values between 0 and 1 [12]. The closer R^2 is to 1 the better is the prediction model. For details concerning the calculation of R^2 see Section 3.3.

Below we present studies, in which fault prediction models were built. For each study we describe the set of metrics used, the metric selection criteria, and the results obtained. The results presented in this section will provide a baseline for evaluation of our models.

A study similar to our one was performed by Zhao *et al.* [22]. The authors compare the applicability of design and code metrics to predict the number of faults. The analyzed system is one release of a large telecommunication system. However, the authors do not say if the modules analyzed are new or modified. The design measures collected are mostly different SDL related metrics (number of SDL diagrams, number of task symbols in SDL descriptions, etc.). Code metrics included the number of lines of code, the number of variables, the number of signals, and the number of if statements. The initial selection of metrics was based on correlation analysis. To build the models the authors used stepwise regression, which additionally eliminated metrics that are not good fault predicators. The authors concluded that both code and design metrics are applicable and give good results. However, the best result was obtained when both types of metrics were included in the same model. R^2 values obtained in this study were 0.63 for the design metrics model, 0.558 for the code metrics model, and 0.68 for the model based on design and code metrics.

The applicability of object-oriented metrics for predicting the number of faults was evaluated by Yu *et al.*[18]. The analyzed system consisted of new classes only. The set of metrics used was largely based on metrics suggested by Chidamber and Kemerer (C&K metrics) [4]. The authors evaluated univariate and multivariate models. The best univariate model was based on Number of Methods per Class (R^2 value = 0.423). The results of univariate regression were used to select metrics for multivariate regression. To be selected the regression model based on the metric had to be significant (t-test) as well as it had to account large proportion of variability of the predicted value. However, in practice, the authors only rejected the variables from the insignificant models. Stepwise regression was used in order to eliminate redundant metrics. Finally six different metrics were included in the proposed regression model. The R^2 statistic of this model was 0.597. The authors also show the model based on all ten metrics they collected which had R^2 value equal to 0.603.

Cartwright and Shepperd [2] present a study in which they predict faults in object oriented system. The metric suite they use consists of some of the object-oriented C&K metrics (depth of inheritance and number of children), some code metrics, and some metrics that are characteristic for the development method employed (Shlaer-Mellor). The authors obtained very high prediction accuracy. Their linear model based only on number of events in the class in state model achieved $R^2 = 0.876$. By adding a variable indicating if there is an inheritance the goodness-of-fit has increased to $R^2 = 0.897$.

A number of other studies were performed to assess the relation between different metrics and fault-proneness. Since they did not use linear regression we do not quote the results obtained in them. They are, however, still interesting since they give indication of which metrics are good predicators of faults. For example, El Emam *et al.* [5] observed the impact of inheritance and coupling on the fault-proneness of the class. The relation between inheritance, coupling, and probability of finding fault in the class was also identified by Briand *et al.* [1].

3. METHODS

3.1 Metrics suite

All metrics that were collected are summarized in Table 1. All measurements were done at the class level. The set of metrics is divided into two groups. One group consists of design metrics (in Table 1 these are the metrics with grey background). The design metrics are mostly metrics that belong to the classic set of object oriented metrics suggested by Chidamber and Kemerer (C&K metrics) [4]. These metrics are normally available before the implementation is done. Most of these metrics can be obtained automatically from design documentation (e.g., can be calculated by CASE tools from UML system design documentation). One exception in this group is the LCOM metric, which can not be calculated automatically from the documentation (though it can be obtained automatically from the code). Manual calculation would significantly increase the cost of obtaining data for the model. In our further discussion we take this issue into account, as metrics that can be obtained automatically are, in practice, much more useful.

Table 1. Metrics collected in the study. The metrics with grey background are design metrics that normally are available before the system is implemented.

Name	Variable	Description
Independent variables		
Coup	Coupling	Number of classes the class is coupled to [4, 8]
NoC	Number of Children	Number of immediate subclasses [4]
Base	Number of Base Classes	Number of immediate base classes [4]
WMC	Weighted Methods per Class	Number of methods defined locally in the class [4]
RFC	Response for Class	Number of methods in the class including inherited ones [4]
DIT	Depth of Inheritance Tree	Maximal depth of the class in the inheritance tree [4, 6]
LCOM	Lack of Cohesion	*"how closely the local methods are related to the local instance variables in the class"[8]*. In the study LCOM was calculated as suggested by Graham [9, 10]
Stmt	Number of statements	Number of statements in the code
StmtExe	Number of executable statements	Number of executable statements in the code
StmtDecl	Number of declarative statements	Number of declarative statements in the code
MaxCyc	Maximum cyclomatic complexity	The highest McCabe complexity of a function within the class
ChgSize	Change Size	Number of new and modified LOC (from previous release)
Dependent variable		
Faults	Number of faults	Number of faults found in the class

The second group are code metrics, which mostly include different size metrics (e.g., number of statements), or metrics describing McCabe cyclomatic complexity (Maximum Cyclomatic Complexity). Since we, in this study, focus on modified code, we have an additional metric describing the size of modification (ChgSize – number of new and modified lines of code in the final system, compared to previous release of the system). ChgSize was obtained by comparing classes from the examined and the previous release of the system. To measure ChgSize we used LOCC application. For details concerning LOCC and the measurement of the size of modification see [21].

The code metrics are not available until the system is implemented. Therefore prediction models that use them can be applied only after the system is fully implemented. All code metrics can be obtained automatically from the code by using appropriate software tools.

3.2 Model building

As stated before, our prediction models should predict the number of faults in modified classes. We want to compare the accuracy of fault prediction models available before the system is implemented with the models available after the system is implemented.

In this study we compare two kinds of models:

- *Univariate models*, which are models based on one variable only.
- *Multivariate models*, which predict the number of faults based on more than one variable.

We have decided to include both univariate and multivariate models because there are different opinions regarding their applicability. Some authors (e.g., [2]) prefer simple univariate models, as such models are supposed to be more stable. Other authors (e.g., [22]) build multivariate models, as such models give potentially higher accuracy.

The models were built using linear regression. Univariate linear regression estimates the value of the dependant variable (number of faults) as a function of the independent variable (code or design metric) [19]:

$$f(x) = a + bx \qquad (1)$$

Multivariate linear regression estimates the value of the dependant variable (number of faults) as a linear combination of independent variables (code and design metrics) [19]:

$$f(x) = a + b_1x_1+b_2x_2+b_3x_3+.....+b_kx_k \qquad (2)$$

To find the metrics for multivariate models we performed a correlation analysis. The correlation co-efficient quantifies the linear correlation between two variables. The value -1 describes perfect negative and +1 described perfect positive correlation. Values close to 0 denote lack of correlation.

For multivariate models we selected variables that were correlated to the number of faults, i.e., with correlation coefficient values not close to 0. For our dataset it turned out that we selected only variables with correlation to the number of faults higher than 0.38. Additionally, the correlation to the number of faults had to be significant at 0.05 level (standard significance level describing 5% chance of rejecting correct hypothesis). In this way we eliminated the metrics that, due to low correlation with faults, can not be considered useful for building prediction models.

To minimize the risk for introducing multicolinearities into our multivariate models we used stepwise regression (see Section 2 for details concerning multicolinearity). Stepwise regression is one of the methods that attempt to build a model on minimal set of variables that explain the variance of the dependant variable. The initial model is built using one variable. Later, a number of iterative steps is performed. In each step the model obtained in the previous step is altered by either adding a new predicator variable to the model or removing a variable from the model. A new variable is added if it improves the ability of the model to explain the dependant variable significantly. The variables that do not contribute the model goodness anymore are removed from the model. For details concerning the stepwise regression see [12].

By using stepwise regression we wanted to get models based on minimal sets of variables. The stepwise regression was applied to the set of metrics that were selected in correlation analysis. It means that not all metrics selected in correlation analysis ended up in the final model.

3.3 Evaluation

Since our goal was to compare the accuracy of prediction of the number of faults in the class we have evaluated our candidate models from that perspective.

The goodness of the model is measured by its "goodness-of-fit". A standard metric applied in measuring "goodness-of-fit" is a co-efficient of determination, R^2. It measures the strength of correlation between the actual and predicted number of faults. The R^2 equation is presented below (3):

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2} \quad (3)$$

where: y_i – actual number of faults, $\hat{y}_i$ -predicted number of faults, $\bar{y}$ - average number of faults.

The practical meaning of R^2 is that it describes the proportion (percentage) of variability of the predicted variable accounted by the model [11]. The higher the R^2 value the better the prediction model fits the actual data. R^2 values range from 0 to 1 [12], where 1 means a perfect model that accounts for all variability of the predicted variable (perfect prediction). R^2 equal to 0 means that the model is useless as prediction model.

Similarly to [22] we have also evaluated the significance of the entire prediction model using F-test [12]. We have selected a 0.05 significance level, i.e., if significance of F-test has a value below 0.05 then the prediction model is significant.

4. RESULTS

As described in Section 3.2 we have begun with building univariate prediction models. The models and their evaluation are presented in Table 2.

From Table 2 it can be noticed that the best univariate model that is available before the system is implemented is the model based on coupling (Coup). It explains about 40% of variability of the dependant variable (number of faults). Also models based on the number of methods (WMC, RFC) are significant and explain some variability of the dependant variable (31%, and 17%, respectively). The model based on the "lack of cohesion" metric, even though significant, accounts for only 2% of variability of dependant variable and, in practice, has no value.

The best univariate model available after the system is implemented is the model based on the size of modification (ChgSize) with R^2 = 55%. The next most promising models are models based on size (Stmt, StmtExe) which account for roughly the same amount of variability (around 40%) as the model based on the best design metric – coupling (Coup).

To select metrics for the multivariate models we have performed a correlation analysis. The results of the correlation analysis are presented in Table 3. Not surprisingly, the findings from the correlation analysis are similar to the observations from building the univariate prediction models. From the metrics available before the system is implemented (marked with italics) as the most promising fault predicators we consider coupling (Coup), weighted method per class (WMC), and response for class (RFC). Base, NOC, and DIT were excluded because of low and insignificant correlation, LCOM because of low correlation with the number of faults. From code metrics, available after the system is implemented, we consider all metrics to be good fault predicators.

Table 2. Univariate prediction models (Nr_of_faults=*bMetric+*intercept*). In the upper part models available before implementation are presented. In the lower part models available only after implementation are presented. Models denoted with italics are the best models from respective groups. R^2 describes goodness-of-fit (coefficient-of-determination), F is the value of F-test, Sig. is the significance of F-test. The models marked with grey background are models that are significant and have R^2 different from 0.**

Metrics available before and after implementation							
Metric	**Base**	***Coup***	**NoC**	**WMC**	**RFC**	**DIT**	**LCOM**
b	-0.032	*0.082*	0.006	0.045	0.021	-0.042	0.006
intercept	0.645	*-0.264*	0.625	-0.217	-0.074	0.653	0.227
R^2	0.00 0	*0.403*	0.000	0.313	0.173	0.000	0.020
F	0.43	*139.666*	0.008	94.306	43.389	0.094	4.179
Sig.	0.836	*0.000*	0.929	0.000	0.000	0.759	0.042
Metrics available only after implementation							
Metric	**Stmt**	**StmtDecl**	**StmtExe**	**MaxCyc**	***ChgSize***		
b	0.002	0.008	0.002	0.030	*0.005*		
intercept	-0.036	-0.026	0.027	0.214	*0.000*		
R^2	0.437	0.278	0.435	0.246	*0.550*		
F	160.594	79.815	159.662	67.681	*252.842*		
Sig.	0.000	0.000	0.000	0.000	*0.000*		

Table 3. Correlation analysis (Spearman correlation co-efficient). Correlations with grey background are NOT significant at 0.05 significance level. The metrics available before the system is implemented are marked with italics.

	Base	*Coup*	*NOC*	*WMC*	*RFC*	Stmt	StmtDecl	StmtExe	MaxCyc	*DIT*	*LCOM*	ChgSize
Base	*1*											
Coup	*0.35*	*1*										
NOC	*-0.13*	*0.06*	*1*									
WMC	*0.23*	*0.76*	*0.18*	*1*								
RFC	*0.63*	*0.68*	*0.07*	*0.82*	*1*							
Stmt	*0.15*	*0.67*	*0.09*	*0.74*	*0.62*	1						
StmtDecl	*0.01*	*0.57*	*0.08*	*0.61*	*0.44*	0.86	1					
StmtExe	*0.25*	*0.72*	*0.10*	*0.79*	*0.70*	0.92	0.64	1				
MaxCyc	*0.21*	*0.65*	*0.09*	*0.65*	*0.56*	0.83	0.51	0.93	1			
DIT	*0.97*	*0.35*	*-0.13*	*0.22*	*0.61*	0.11	-0.01	0.22	0.12	*1*		
LCOM	*-0.08*	*0.3*	*0.18*	*0.37*	*0.15*	0.20	0.38	0.11	0.07	*-0.08*	*1*	
Chg Size	*0.07*	*0.48*	*0.02*	*0.47*	*0.40*	0.68	0.7	0.50	0.42	*0.04*	*0.28*	1
Faults	*0.00*	*0.43*	*0.02*	*0.47*	*0.41*	0.52	0.48	0.48	0.38	*-0.01*	*0.15*	0.6

We have built following multivariate models:

- *Design* – a model built using selected design metrics (Coup, WMC, RFC). This model is available before the system is implemented
- *Design+Code* – a model that is available after the system implemented. As model input we use selected design metrics and all code metrics
- *Design+Code-ChgSize* – same as Design+Code but without ChgSize metric
- *Code* – a model based on code metrics only.

Some of the models were introduced for special reasons. The Design+*Code-ChgSize* is a model available after the system is implemented, in which we excluded the *ChgSize* metric. There are two reasons for introducing this model. First, to collect *ChgSize* data it is not enough to measure the final code or design. We need historical information about previous class release, which makes data collection and processing more complex. The second reason is that ChgSize is a metric of a different type than all other metrics. It describes the characteristic of the modification, not the characteristic of the final product. By comparing the accuracy of models with and without this metric we should get some indication to what extent metrics describing the characteristics of modifications are useful.

The reason for introducing the *Code* model was to enable the comparison of our findings with findings of other authors that usually tend to introduce such a model in their published work (see Section 2 for examples). In this way we get some indication to what extend our findings are typical.

We had some concerns regarding the relatively high cost of collecting the LCOM metric (see Section 3.1). Because LCOM is not considered useful for building prediction model our concerns are not valid anymore.

To build the prediction models we have used stepwise regression. As we mentioned before (see Section 3.2) this method attempts to build a model using minimal set of variables. Therefore the final models use subsets of metrics that were used as input for model building. The results of model building and evaluation are summarised in Table 4.

The model that is available before the system is implemented (*Design*, marked in Table 4 with grey background) explains 43% of variability of the dependant variable. The models available after the system is developed are better, they explain up to 58.5% of fault variability. When building *Design+Code* model the stepwise regression procedure selected only code metrics. Therefore, *Design+Code* and *Code* models are exactly the same. Excluding ChgSize metric affected significantly the accuracy of fault prediction. The "after implementation" model based on all metrics but ChgSize (*Design+Code-ChgSize*) is only a little bit better than models available before the implementation. It explains about 48% of fault variability ("before implementation" model explains up to 44%).

Table 4. Prediction models obtained using stepwise regression. R^2 describes goodness-of-fit (values closer to 1 indicate better fit), Sig. is significance level of F-test. The model with grey background is available before the system is implemented. *Design+Code* and *Code* models are the same because in stepwise regression only code metrics were selected to the model.

Model	Equation	R^2	F	Sig.
Design	Faults = 0.062*Coup+0.018*WMC-0.384	0.429	77.412	0.0
Design+Code	Faults = 0.004*ChgSize+0.001*StmtExe-0.002*StmtDec+0.008	0.585	96.412	0.0
Design+Code-ChgSize	Faults = 0.001*Stmt+0.04*Coup-0.227	0.474	92.796	0.0
Code	Faults = 0.004*ChgSize+0.001*StmtExe-0.002*StmtDec+0.008	0.585	96.412	0.0

5. DISCUSSION

The goal of this paper was to compare the accuracy of fault prediction models based on metrics available before implementation (design metrics) and metrics available after the system is implemented. By looking at Table 4 we can see that models based on metrics available after the system is implemented provide more accurate prediction (R^2 equal to about 58%) compared to models available early in design process (R^2 = 43%). This means that prediction made after the system is implemented is about 34% more accurate.

A closer analysis reveals that the metric that contributes most to the accuracy of the models that are available after the implementation is the size of modification (ChgSize). It alone is able to explain 55% of the fault variability (see univariate model based on ChgSize in Table 2). In our study this metric does not have any corresponding metric among the design metrics. We can potentially think of design metrics based on similar concept, e.g., by applying similar ideas to coupling (amount of new and modified coupling relations) we could potentially obtain some design level metric describing the change. To our best knowledge the collection of such metrics would, given available tools, be rather complex. However, it is an interesting research direction. From our findings, it seems that the characteristics of the modification affect fault-proneness more than the final characteristics of the class. If we exclude the ChgSize metric we can see that prediction accuracy of design and code metrics is roughly the same. It would be similar to the findings of other researchers in the area (see Section 2 for details concerning related work). One difference is that in our work code metrics tend to give slightly better results compared to design metrics, while, e.g., [22] reported opposite findings. In [22], however, different design metrics were used as well as it is not clear if the code there was new or modified.

When it comes to our prediction accuracy, there are studies (e.g., [2, 22]) that report better accuracies. However, when it comes to prediction using object oriented metrics (C&K) the results obtained by other researchers (e.g., [18], see Section 2 for values) are very similar. It can be an indication that the accuracy of our models is typical and our results are rather reliable.

An interesting finding is that the accuracy of univariate models is not much lower compared to the accuracy of multivariate models. As many authors claim [2, 7] the simple models are preferable since they do not suffer from the risk of multicolinearity (see Section 2 for details). Even our multivariate models, despite the fact that they consist of rather modest number of metrics, have some signs of multicolinearity. For example, in Design+Code model (see Table 4) the sign before StmtDecl is negative (i.e., increasing the number of declarative statements decreases the amount of faults in the class), while from Table 3 we clearly see that the correlation between Faults and StmtDecl is positive, which suggests something opposite.

A great advantage of our models is that they can be applied automatically. All metrics can be extracted from the code by appropriate software tools in the matter of minutes. Design metrics can be also extracted from design documentation (e.g., UML diagrams). As we have mentioned before, the only problem is LCOM metric that would require manual calculation. However, as it can be noticed in Table 2 and Table 3, LCOM is not particularly useful for predicting the number of faults. Therefore there is no point in collecting this metric before the system is implemented.

When discussing our findings we must state that a big threat to validity of our study is that we have built and evaluated the model using the same data. Therefore we have not tested the most important feature of the prediction models – how well they predict faults when they are applied to other project/release. Unfortunately, when performing the study, we had no access to similar data from other project or other release of the same project. We believe that our findings can be generalized, mostly based on similarities of our findings with findings of other researchers. However we have not performed any empirical validation of our models. We plan to perform such validation in future. Therefore, our current study has to be considered to be more of a feasibility study.

6. CONCLUSIONS

The goal of the study was to compare the accuracy of fault prediction models available before and after the system is implemented. We have built and evaluated a number of fault prediction models based on the data collected from a large telecommunication system. The models were built using linear regression. We have built univariate and multivariate models. The models were evaluated using a standard metric (coefficient of determination – R^2) that describes the percentage of variability of faults accounted for by the model.

We have found that prediction models that include metrics available after the system is implemented provide about 34% more accurate prediction. It concerns both univariate and multivariate fault prediction models. The models based on metrics available after the implementation were able to explain about 58% of fault data variability. Prediction model based on metrics available before the system is implemented explained about 43% of fault variability.

We have discovered that the higher accuracy of models available after the system is implemented is caused by the code metric ChgSize, which describes the size of modification. Without this metric the fault prediction accuracy before and after the implementation is roughly the same, both in case of univariate and multivariate models. In this study we did not have any corresponding design metric that would characterize the modification. It seems like a promising way forward to look for such a metric and test its applicability for early fault prediction in modified code.

In this study we have also found that the gain connected with building multivariate models over univariate models is rather limited for modified code and our set of metrics. The univariate models were about 5% less accurate compared to their multivariate counterparts, which we do not see as a major difference. Since univariate models are more stable, as they are not prone to multicolinearity, we suggest using them for predicting the number of faults in modified code.

7. ACKNOWLEDGMENTS

The authors would like to thank Ericsson for providing us with the data for the study and The Collaborative Software Development Laboratory, University of Hawaii, USA (http://csdl.ics.hawaii.edu/) for LOCC application.

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

8. REFERENCES

[1] L. C. Briand, J. Wust, J. W. Daly, and D. V. Porter, "Exploring the relationship between design measures and software quality in object-oriented systems", *The Journal of Systems and Software*, vol. 51, 2000, pp. 245-273.

[2] M. Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system", *IEEE Transactions on Software Engineering*, vol. 26, 2000, pp. 786-796.

[3] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis", *IEEE Transactions on Software Engineering*, vol. 24, 1998, pp. 629-639.

[4] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol. 20, 1994, pp. 476-494.

[5] K. El Emam, W. L. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics", *The Journal of Systems and Software*, vol. 56, 2001, pp. 63-75.

[6] N. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system", *IEEE Transactions on Software Engineering*, vol. 26, 2000, pp. 797-814.

[7] N. E. Fenton and M. Neil, "A critique of software defect prediction models", *IEEE Transactions on Software Engineering*, vol. 25, 1999, pp. 675-689.

[8] N. E. Fenton and S. L. Pfleeger, *Software metrics: a rigorous and practical approach*, 2. ed. London; Boston: PWS, 1997.

[9] I. Graham, *Migrating to object technology*. Wokingham, England; Reading, Mass.: Addison-Wesley Pub. Co., 1995.

[10] B. Henderson-Sellers, L. L. Constantine, and I. M. Graham, "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)", *Object Oriented Systems*, vol. 3, 1996, pp. 143-158.

[11] G. Keppel, *Design and analysis: a researcher's handbook*, 4. ed. Upper Saddle River, N.J.: Prentice Hall, 2004.

[12] J. S. Milton and J. C. Arnold, *Introduction to probability and statistics: principles and applications for engineering and the computing sciences*, 4. ed. Boston: McGraw-Hill, 2003.

[13] A. P. Nikora and J. C. Munson, "Developing fault predictors for evolving software systems", *Proceedings of The Ninth International Software Metrics Symposium*, 2003, pp. 338-349.

[14] N. Ohlsson, A. C. Eriksson, and M. Helander, "Early Risk-Management by Identification of Fault-prone Modules", *Empirical Software Engineering*, vol. 2, 1997, pp. 166-173.

[15] N. Ohlsson, M. Zhao, and M. Helander, "Application of multivariate analysis for software fault prediction", *Software Quality Journal*, vol. 7, 1998, pp. 51-66.

[16] M. Pighin and A. Marzona, "An empirical analysis of fault persistence through software releases", *Proceedings of the International Symposium on Empirical Software Engineering*, 2003, pp. 206-212.

[17] M. Pighin and A. Marzona, "Reducing Corrective Maintenance Effort Considering Module's History", *Proc. of Ninth European Conference on Software Maintenance and Reengineering*, 2005, pp. 232-235.

[18] Y. Ping, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study", *Proc. of The Sixth European Conference on Software Maintenance and Reengineering*, 2002, pp. 99-107.

[19] D. G. Rees, *Essential statistics*, 3rd ed. London; New York: Chapman & Hall, 1995.

[20] I. Sommerville, *Software engineering*, 7th ed. Boston, Mass.: Addison-Wesley, 2004.

[21] The Collaborative Software Development Laboratory, University of Hawaii, USA;LOCC Project Homepage, http://csdl.ics.hawaii.edu/Tools/LOCC/;2005

[22] 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, 1998, pp. 801-809.