Increasing the Efficiency of Fault Detection 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

Many software systems are developed in a number of consecutive releases. Each new release does not only add new code but also modifies already existing one. In this study we have shown that the modified code can be an important source of faults. The faults are widely recognized as one of the major cost drivers in software projects. Therefore we look for methods of improving fault detection in the modified code. We suggest and evaluate a number of prediction models for increasing the efficiency of fault detection. We evaluate them against the theoretical best model, a simple model based on size, as well as against analyzing the code in a random order (not using any model). We find that using our models provides a significant improvement both over not using any model at all and using the simple model based on the class size. The gain offered by the models corresponds to 30% to 60% of the theoretical maximum.

1. Introduction

Development of software for telecommunications requires meeting many contradicting goals. There are high expectations concerning both system functionality and quality. On the other hand, the competitiveness of the telecom market puts a large pressure on decreasing the development cost. One element that contributes significantly to the cost of software development are faults. Finding and fixing them is a very expensive activity [26]. In telecommunications fault removal activities can account for a significant part of the project budget, e.g., in [3] 45% of the project resources were devoted to testing and simulation.

Typically telecommunication systems have a number of releases. At first the basic system is developed. Later perfective maintenance activities lead to newer versions of the system. Since a new release usually introduces a significant amount of new functionality it often results in major changes introduced to the current system. Such code modifications are an important source of faults [22].

A well known fact concerning faults is that a majority of faults can be found in a minority of code (e.g., 60% of faults can be found in 20% of the modules [7, 20]). A prediction model that identifies the most fault-prone code can bring significant savings on project cost. It makes it possible to focus fault detection activities on the code that is most fault-prone, thus decreasing the cost of finding faults by increasing the *fault detection efficiency* (i.e., the number of faults found/amount of code analyzed). Such fault prediction models are usually based on different characteristics of the software, e.g., design or code metrics (e.g., [6, 29]) or historical information about the code (e.g., [22, 23]).

This paper presents our experiences from building a fault prediction model based on data from one release of a large telecommunication system developed by Ericsson. The system comprised about 600 classes (250 KLOC). About 40% of the code was new compared to previous release. 65% of the new code was introduced as modification to 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 a prediction model for modified classes, as faults in those classes contribute most to the total number of faults in the system. The model is based on metrics collected after the system was developed. The goal for building the model is to increase the efficiency of fault detection. In the study we have made the assumption that the cost of finding faults in the class is directly proportional to the size of the class (the same assumption as in [1, 2]). Therefore the model should be able to predict classes with high fault density, since looking for faults in classes with high fault density is the most cost-efficient.

One possible application of the model is when there are limited resources to perform in-depth code

analysis. Given the percentage of code that we are able to analyze (which is limited by given resources) the model should pin-point classes (that account for given percentage of the code), analyzing which would result in detecting the largest number of faults.

The rest of the paper is structured as follows: in Section 2 we present the work that has been done by the others in the area of fault prediction, Section 3 describes methods we have used for model building and evaluation. Section 4 presents the results we have obtained. 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 fault prediction models that can be found in the literature usually aim at predicting [17]:

- Number of faults – these models predict the number of faults in the code unit. Examples of such models can be found in [3, 4, 19, 24, 29].
- Fault-proneness – these models predict if the code unit (e.g. a class) has faults. Examples of such models can be found in [1, 6, 10, 16].

The models that have number of faults as dependant variable (variable that we predict) are often referred as quality prediction models, while those that predict fault-proneness are called classification models [17]. Our model is clearly a quality prediction model since classifying classes as fault-prone and not fault-prone would not let us estimate their fault-density.

Usually the prediction model construction starts with selecting independent variables (variables that are used to predict dependant variable). The most common candidates are different code metrics (e.g., [14, 23, 29]) or variations of C&K [5] object oriented metrics (e.g., [1, 6, 29]). There are also studies that take historical information about code fault-proneness into account (e.g., [22, 23]). The initial set of independent variables is often large. A common assumption is that models based on a large number of variables are less robust and have lower practical value (more metrics have to be collected) [3, 8]. Therefore the first step often involves reduction of the number of metrics. A commonly used method in dataset reduction is correlation analysis ([3, 6, 29]). It is usually used for two purposes. One is to detect highly correlated metrics. Such metrics, to a large extent, can measure the same thing (e.g., number of code lines and number of statements are usually highly correlated because both measure size). Introducing them into the model causes a risk for multicolinearity [3]. Multicolinearity is especially risky when regression models are built. It leads to "*unstable coefficients, misleading statistical tests, and unexpected coefficient signs*"[8]. Other usage of correlation coefficients includes selection of independent variables to predict faults (e.g., [20, 29]). Only the metrics that are correlated with faults are good fault predicators.

The methods for building models range from uni- and multivariate linear regression (e.g., [3, 4, 19, 21, 24, 29]) and logistic regression (e.g., [1, 6, 10, 16]) through regression trees (e.g., [14, 15]) to neural networks (e.g., [17, 27]).

In [29] Zhao *et al.* compare the applicability of design and code metrics to predict the number of faults. They conclude that both types of metrics are applicable and give good results. However, the best result was obtained when both types of metrics were included in the same model. The number of variables in the code and the number of internal signals in the module were identified as the most promising fault predicators. El Emam *et al.* [6] 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]. Cartwright and Shepperd also observed [3] higher fault-densities in classes with high inheritance. They observed that the number of events per class works well as a single predicator variable. They also noticed that the number of events is highly correlated with LOC size measure.

When it comes to evaluation, most classification models are evaluated against the percentage of correctly classified classes. Briand *et al.* [1] noticed that such an evaluation may have low practical value. Even though the model points to a minority of classes, these classes can potentially account for a majority of the code. The prediction models used for estimating number of faults are usually evaluated against their "goodness of fit", e.g., using R^2 statistic. Therefore, as we see it, there is a lack of studies evaluating prediction models from the perspective of gain, in terms of cost reduction, that can be expected from applying the fault prediction model.

3. Methods

3.1 Metrics suite

All metrics that were collected are summarized in Table 1. Based on the experiences described in [29] we decided to base our prediction model on both code and design metrics. All measurements were done at the class level. The design metrics are mostly metrics that belong to the classic set of object oriented metrics suggested by Chidamber and Kemerer (C&K metrics) [5]. The code metrics are different size metrics (e.g.,

Table 1. Metrics collected in the study

Name	Variable	Description
Independent metrics		
Coup	Coupling	Number of classes the class is coupled to [5, 9]
NoC	Number of Children	Number of immediate subclasses [5]
Base	Number of Base Classes	Number of immediate base classes [5]
WMC	Weighted Methods per Class	Number of methods defined locally in the class [5]
RFC	Response for Class	Number of methods in the class including inherited ones[5]
DIT	Depth of Inheritance Tree	Maximal depth of the class in the inheritance tree [5, 7]
LCOM	Lack of Cohesion	*"how closely the local methods are related to the local instance variables in the class"* [9]. In the study LCOM was calculated as suggested by Graham [11, 12]
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
Comment	Number of comments lines	Number of lines containing comments
MaxCyc	Maximum cyclomatic complexity	The highest McCabe complexity of a function from the class
ChgSize	Change Size	Number of new and modified LOC (from previous release)
CtC	Ratio Comment to Code	Ratio of comment lines to code lines
Dependent variables		
Faults	Number of faults	Number of faults found in the class
FaultDensity	Fault density	Fault density of the class

number of statements), metrics describing McCabe cyclomatic complexity (Maximum Cyclomatic Complexity) as well as metrics describing the size of modification (Change size – number of new and modified lines of code in the final system, compared to previous release). All measurements mentioned in this study can be obtained automatically from the code. For each class we have collected information about the number of faults that were found in it.

3.2 Model building

The goal of the model is to increase the efficiency of fault detection. We assume that the cost of performing fault detection is directly proportional to the size of the class. Therefore, the prediction model should identify the classes with the highest fault density. Fault detection in such classes is the most efficient because it requires least code to be analysed to find a fault. Class analysis according to the model means that fault detection activities are performed on classes in the order of their fault density predicted by the model. As we see it, the fault density can be predicted in two ways:

- by predicting fault density (Faults/Stmt) – fault density is a dependant variable in the model.
- by predicting the number of faults (Faults) and dividing the predicted number of faults by real class size (Stmt) – Faults are predicted by the model, while size (Stmt) is measured.

In our study we evaluate both approaches. Even though they seem to predict the same thing the prediction accuracy, given our set of metrics and our method of building models (regression), may be different for both. Linear regression attempts to predict the dependent variable as linear combination of independent variables. It may turn out that, e.g., linear combination of our metrics predicts fault density much more accurately than the number of faults.

We evaluated six prediction models, three predicting the fault-density and three predicting the number of faults. The models were built using:

- single metric – a model based on single best fault (fault-density) predicator
- selected metrics – a model based on a set of the best fault (fault-density) predicators
- all metrics – a model based on all metrics collected

To find the single and the selected metrics we performed a correlation analysis. Since it turned out that our data were not normally distributed we used Spearman correlation co-efficient which is not dependent on normality assumption [28]. As selected metrics we chose those that were correlated to independent metrics, i.e., with correlation coefficient values not close to 0. Additionally, the selected metrics' correlations to dependent variables 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 number of faults and fault-density, can not be considered useful for building prediction models.

As we said before, the model based on all metrics may be prone to overfitting and instability between releases (may be a poor prediction model) because of multicolinearities [3, 8]. We, however, decided to introduce such models because they should be able to fit current data best. Therefore they can be considered as some kind of measurement of how good the model based on our set of metrics can be.

The models were built using stepwise multivariate linear regression. Linear regression estimates value of dependant variable (number of faults or fault-density) using linear combination of independent variables (code and design metrics) [25]:

$$f(x) = a + b_1x_1+b_2x_2+b_3x_3+.....+b_kx_k \qquad (1)$$

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. Other methods of that kind are forward (backward) regression. In these methods variables are added (removed) to the model until adding (removing) the next one does not give any benefit (does change models ability to predict a dependant variable) [18]. We selected stepwise regression because, compared to forward regression, it additionally excludes variables that do not contribute to the model anymore [18]. By using stepwise regression we hoped to get models based on minimal sets of variables. The stepwise regression was used on the previously defined sets of metrics (All, Selected) so final models were build on subsets of the previously defined sets of metrics, i.e., building model on all metrics did not mean that all metrics were used in the model but that all metrics were used as an input to stepwise regression.

3.3 Model evaluation

Since the goal of the model was to increase the efficiency of fault detection we evaluated our candidate models from that perspective. The goodness of the model is measured by the amount of code necessary to analyze in order to detect a certain number of faults, i.e., a model is better if by following it we are able to detect more faults by analyzing the same amount of code compared to another model. Therefore, for each model, we plot a diagram describing the percentage of faults detected against the percentage of code that had to be analyzed to detect them.

We introduce three models against which we benchmark our models:

- *Random model* – the model describing completely random search for faults
- *Best model* – the theoretical model that makes only the right choices about which class to analyze
- *Size model* – a common (mis)conception [8] is bigger classes tend to have more faults and higher fault densities. Therefore we introduce a model in which the classes were analyzed based on their size (bigger classes are analyzed first)

A comparison with the Random model gives us an indication to what extent following our model is better than not following any model at all. The Best model gives us an indication of how good the model can be at all, and how far we are from being perfect. The Size model might be often encountered in real life situations because of its simplicity as well as because many models suggested in literature actually tend to correlate with size [8]. By including this model we can evaluate it against our criteria of efficiency improvement as well as compare our models with it.

4. Results

4.1 Model building

As described in Section 3.2 we began the model building with a correlation analysis. The results of the correlation analysis are presented in Table 2. The main purpose of the correlation analysis was to identify metrics that are the best single predicators of the number of faults and the fault-density (we looked for single metrics with the highest correlation to the number of faults and the fault density). From Table 2 it can be noticed that ChgSize is the best predicator for both values (correlation values in bold in Table 2). Therefore it was selected to build the prediction models for the number of faults and the fault-density based on one metric.

The second reason for performing correlation analysis was to eliminate the metrics that can not be considered useful for building prediction models (see Section 3.2 for details). Remaining metrics were used to build the model based on “selected metrics”. It turned out that we removed the same metrics for the model that predicts the number of faults and the model that predicts fault-density. We have decided not to use the following metrics in “selected metrics” models:

Table 2. Correlation analysis (Spearman correlation co-efficient). Correlations with grey background are NOT significant at 0.05 significance level. Correlation of the best individual predicator of fault number and fault density in bold.

	Base	Coup	NOC	WMC	RFC	Com ment	Stmt	Stmt Decl	Stmt Exe	Max Cyc	DIT	LC OM	CtC	Chg Size
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									
Comment	0.21	0.73	0.05	0.80	0.68	1								
Stmt	0.15	0.67	0.09	0.74	0.62	0.84	1							
StmtDecl	0.01	0.57	0.08	0.61	0.44	0.73	0.86	1						
StmtExe	0.25	0.72	0.10	0.79	0.70	0.83	0.92	0.64	1					
MaxCyc	0.21	0.65	0.09	0.65	0.56	0.73	0.83	0.51	0.93	1				
DIT	0.97	0.35	-0.13	0.22	0.61	0.20	0.11	-0.01	0.22	0.12	1			
LCOM	-0.08	0.3	0.18	0.37	0.15	0.32	0.20	0.38	0.11	0.07	-0.08	1		
CtC	0.18	-0.01	-0.06	-0.02	0.07	0.12	-0.36	-0.34	-0.24	-0.26	0.21	0.06	1	
ChgSize	0.07	0.48	0.02	0.47	0.40	0.59	0.68	0.7	0.50	0.42	0.04	0.28	-0.25	1
Faults	0.00	0.43	0.02	0.47	0.41	0.54	0.52	0.48	0.48	0.38	-0.01	0.15	-0.1	**0.6**
Fault Density	-0.03	0.35	0.01	0.39	0.32	0.46	0.42	0.4	0.36	0.29	-0.04	0.15	-0.05	**0.53**

- Base, NOC, CtC, DIT – due to their low correlation with faults and fault density and low significance of the correlation
- LCOM – due to low correlation with faults and fault-density
- Comment – due to unsure meaning of this metric and its correlation to size

We were quite surprised with the high positive correlation between Comment and Faults. There are some possible explanations of that phenomenon, like considering the number of comments as a measure of human perceived complexity. We excluded this metric, because it is difficult to assure that "commenting style" is maintained between the projects (no explicit guidelines concerning it in the analyzed project). Therefore it is difficult to say if prediction model based on Comments would be stable across releases.

In the study we have built six different prediction models. They are summarized in Table 3. The models were built using stepwise regression. The significance of each model's coefficient was checked using t-test. The hypothesis tested was that the coefficient could have value 0, which would imply a lack of relationship between the independent and the dependant variable (and therefore would make the model the best, but not meaningful, mathematical relation between both variables) [25]. It turned out that all coefficients were significant at 0.05 level. The significance of the entire models was tested using F-test. Goodness-of-fit of each model was assessed using R^2 statistic (correlation between the actual values and the predicted values, values between 0 and 1, where 1 is perfect fit – the best prediction). The models are presented in Table 4.

4.2 Model evaluation

As it can be noticed in Table 4 all models are significant according to the F-test. The goodness-of-fit values (R^2) are better for the models predicting the number of faults compared to those predicting the fault-densities. Apparently, given our set of metrics, the number of faults is easier to predict than the fault-density. As we predicted in advance (see Section 3.2) the models based on all metrics (AllNumber and AllDensity) have a better fit compared to their counterparts based on a limited number of metrics (see R^2 values in Table 4). They may, however, suffer from multicolinearity problem – e.g., according to

Table 3. Models summary. Single metric: ChgSize. Selected metrics: Coup, WMC, RFC, Stmt, StmtDecl, StmtExe, MaxCyc, ChgSize

Name	Based on	Predicts
AllNumber	All metrics	Number of faults
SelNumber	Selected metrics	Number of faults
SingleNumber	Single metric	Number of faults
AllDensity	All metrics	Fault density
SelDensity	Selected metrics	Fault density
SingleDensity	Single metric	Fault density

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

Model	Equation	R^2	F	Sig.
AllNumber	Faults = 0.004*Comment + 0.003*ChgSize - 0.677*Base+ 0.276*Ctc - 0.003*StmtDecl -0.005*LCOM + 0.010*RFC - 0.001*StmtExe + 0.089	0.752	75.944	0.0
SelNumber	Faults = 0.004*ChgSize + 0.001*StmtExe - 0.002*StmtDecl + 0.008	0.585	96.412	0.0
SingleNumber	Faults = 0.005*ChgSize	0.550	252.84	0.0
AllDensity	FaultDensity = 54.040*CtC + 0.115*ChgSize - 42.431*DIT - 0.096*Stmt + 3.036*Coup + 0.670*RFC - 19.685	0.479	30.997	0.0
SelDensity	FaultDensity=0.184*ChgSize-0.138*Stmt+2.429*Coup+1.057*WMC+2.748	0.280	19.815	0.0
SingleDensity	FaultDensity=0.081*ChgSize + 12.739	0.058	12.742	0.0

AllNumber model the number of faults increases with Comments and decreases with StmtExe which is difficult to explain since both StmtExe and Comments are positively correlated with the number of faults.

All models were evaluated from the perspective of fault detection efficiency. We used the model as an indicator of the order in which classes should be analyzed. For the models that predict the fault-density (AllDensity, SelDensity, SingleDensity) we ordered classes according to the output of the model, so that we analyze classes with highest predicted fault-density first. In the models that predict the number of faults (AllNumber, SelNumber, SingleDensity) the predicted number of faults was divided by class size (Stmt). This partially predicted density measure was used to select classes for analysis.

To benchmark our models we have included three reference models in the evaluation. The Random model corresponds to analysis in which classes are picked randomly, the Best model describes a perfect model that makes only the right choices. The Size model is a model in which the biggest classes are analyzed first (see Section 3.3 for details concerning reference models). The evaluation of the reference models is presented in Figure 1. The reference models are present in all our future diagrams (always dashed lines).

Two conclusions can be drawn from the evaluation of the reference models presented in Figure 1. The first one is that by comparing the Best and the Random model we can see that there is a large room for improvement that can be filled using a fault prediction model. The second conclusion is that the Size model does not fill that room very well. It is slightly better than the Random model for first 30% of code. Later it becomes worse than random selection of classes.

Another finding is that our dataset seems to support the 60/20 rule [7, 20] stating that 60% of the faults can be found in 20% of the code.

The evaluation of fault detection efficiency improvement gained by using the models that predict the number of faults (AllNumber, SelNumber, SingleNumber) is presented in Figure 2. As it can be noticed all three models present an improvement over both the Random model as well as the simple Size model. As we have predicted the model based on all variables (AllNumber) is the best. However, the other two models (SingleNumber, SelNumber) are not doing significantly worse. Given our previous concerns regarding stability of the models based on all variables (see Section 3.2 for details) we can safely say that, if the models based on limited number of variables prove to be more stable over releases, then using them also improves the fault detection efficiency. The gains from using each of the models are presented in Figure 3.

As can be noticed in Figure 3 the difference in the efficiency between SingleNumber and SelNumber is

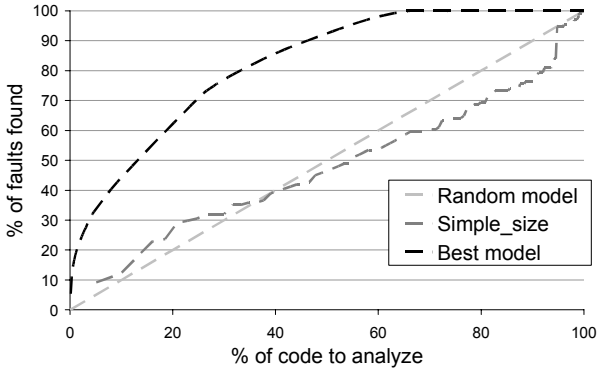

Figure 1. Reference models - evaluation

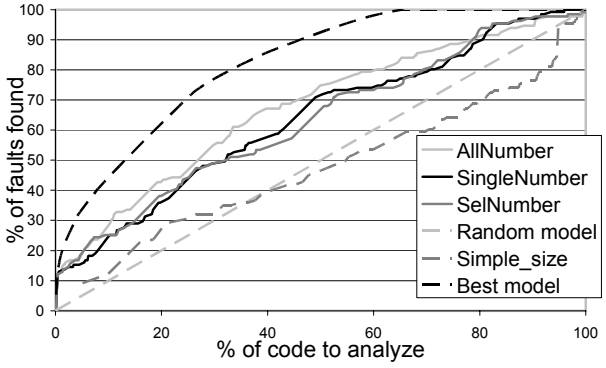

Figure 2. Fault prediction models - evaluation

minimal. Both models offer about 40%-50% of the improvement of the Best model (in Figure 3 the SingleNumber and SelNumber lines lie usually about half-way between the Best model line and the "% of code to analyze" axis). The model based on all variables (AllNumber) seems to constantly offer about 50%-60% of the improvement that could be expected from the perfect model. All models represent significant improvement over the Random model.

The same evaluation was performed for the models that predict fault-density (AllDensity, SelDensity, SingleDensity), The results of that evaluation are presented in Figure 4.

As before, the model based on all variables seems to be the best, at least when we analyze up to 60% of the code. However, one interesting finding is that the model based on a single variable (SingleDensity), on average, performs not much worse than the model based on all variables and outperforms the model based on selected variables (SelDensity). The analysis of gains from using each of the fault-density prediction models are presented in Figure 5. Between 25% and 70% of the code the SingleDensity model provides about 30% of improvement of the Best model. SelDensity model is fairly stable in providing about 20% of the Best model's improvement over random model. The model based on all variables (AllDensity) provides between 40% and 50% of the efficiency improvement offered by the Best model.

5. Discussion

5.1 Findings

The results obtained in our study are promising. All our models (AllNumber, SelNumber, SingleNumber, AllDensity, SelDensity, SingleDensity) represent an improvement compared to the Random model. It means that, when focusing fault detection efforts on the part of the code only, more faults would be detected

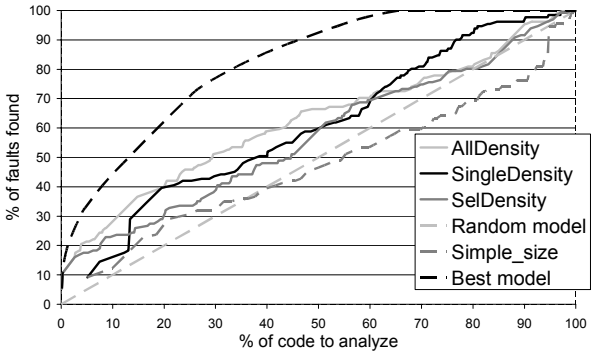

Figure 4. Fault-density prediction models - evaluation

when using the model compared to analysing the classes in random order. The gain from following the model corresponds to 30% to 60% of the theoretical maximum improvement possible. The exact value of the gain depends on the selected model and percentage of code analysed. The best results were obtained when using models based on all variables. Unfortunately, such models most often suffer from multicolinearity, which makes them poor prediction models [3, 8]. However, our models that contain a limited number of variables still provide an improvement over both Random and Size models. The capabilities of the models based on all variables can be considered an upper limitation of building models using our set of metrics and linear regression. As we can see, on average, we can be only half as good as the perfect model. Therefore there is still some room for improvement, e.g., by introducing new metrics, deriving new variables from our metrics and using prediction methods other than linear regression.

The comparison of goodness-of-fit (R^2 values) of models predicting number of faults and predicting fault-density indicates that, given our set of metrics,

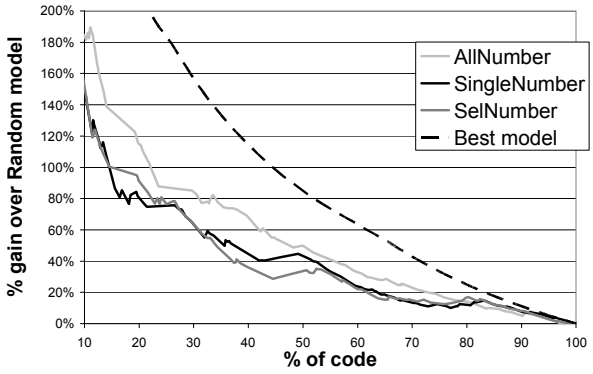

Figure 3. Fault prediction models – gain over the random model

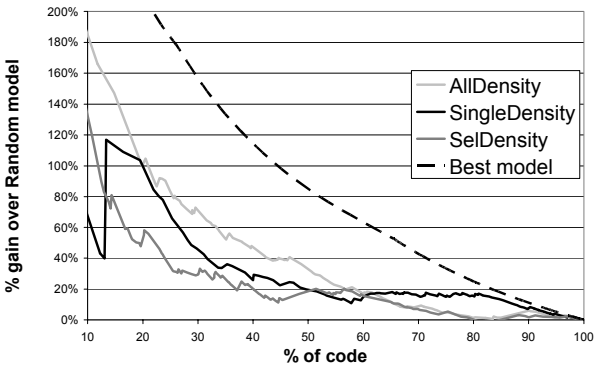

Figure 5. Fault-density prediction models – gain over the random model

the prediction of number of faults is much more accurate. The R^2 values of number of faults prediction models (0.752, 0.585, 0.550 - see Table 4 for details) are typical R^2 values for prediction models that can be found in literature. For example, [19, 29] report following R^2 values: 0.558, 0.611, 0.63, 0.68. The R^2 values for density prediction models are much lower (0.479, 0.280, 0.058 – see Table 4 for details), which practically disqualifies them as prediction models. However, when comparing the performance of fault and fault-density prediction models when it comes to fault detection efficiency, the difference between both types of models was not so huge. One explanation could be that, even though the density prediction models predict inaccurate values, they maintain order, i.e., they predict higher values for classes with higher fault densities. Since it is the order that counts in our case, the result turns out to be rather good. It is, however, very difficult to say if this characteristic will be stable over system releases.

An interesting and promising finding is that models based on a single variable (SingleNumber, SingleDensity) perform at least as good as the models based on selected variables (SelNumber, SelDensity). Such models based on low number of variables are preferable because they are usually more stable over releases [3, 8]. The comparison of the performance of SingleNumber and SingleDensity is presented in Figure 6. Because of better overall performance (see Figure 6) as well as better goodness of fit (see Table 4) we can say that from models based on single variable the SingleNumber is the preferable one. The SingleNumber model is based on ChgSize variable and predicts the number of faults (see Table 4). When applying it, we divide the predicted number of faults by class size to obtain a density metric. Therefore we can say that change density turned out to be the most promising single fault predicator for modified code.

Apart from the findings concerning our models we have also made some more general observations that can contribute to the body of knowledge in metrics research. One of them is yet another empirical evidence for 60/20 rule [7, 20], stating that 60% of faults can be found in 20% of modules. Our results show that it is also truth for the code, 20% of code contains 60% of faults. Another finding is that the size of the class is a poor predicator of fault-density in modified classes (see Section 4.2 for details). However, we have also found that the size of the change is a good predictor for fault-density and number of faults.

A surprising finding was high positive correlation between number of comments and faults. As we said, one way of interpreting it is to consider comments a measure of human perceived complexity, which could

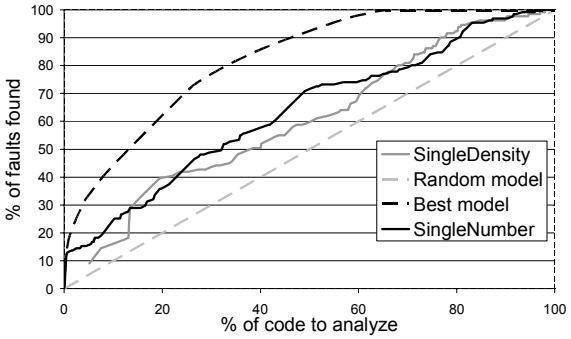

Figure 6. Prediction models based on single variable - comparison

explain the positive correlation. It might be interesting to see if such a correlation is a characteristic of our dataset only or if it is present in other datasets as well. In the future studies we plan to investigate this issue.

It can be noticed that all our models use code metrics extensively. These metrics are available after the system was developed. It might be interesting to investigate what we can expect from models based on design metrics only. Such metrics are available before the system is developed, which gives an opportunity of taking some preventive measures on especially fault prone classes (e.g., more careful development, increased number of code reviews, assignment of such classes to more experience developers).

5.2 Validity

As suggested in [28] we distinguish between four types of validity: internal, external, construct and conclusion validity.

The *internal validity* "concerns the causal effect, if the measured effect is due to changes caused by the researcher or due to some other unknown cause" [13]. Since our study is mostly based on correlations, by definition we can not claim the causal relationship between our dependant and independent variables. However, it is also not our ambition to claim that. There can be (and probably is) an underlying third factor that demonstrates itself in both dependent and independent variables and therefore it is possible to predict one of them using another. Because of that, by finding correlations we are able to build a useful prediction model.

The *external validity* concerns the possibility of generalising the findings. The study was performed on a system, which is representative for systems of its class (i.e., telecommunication systems). The system is rather large (250 KLOC, 40% of the code was new in the system release we have studied). One 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 project. Therefore, our study can be considered to be a feasibility study.

The *construct validity* "reflects our ability to measure what we are interested in measuring" [13]. One thing that may be worth discussing is the assumption that an effort connected with the fault detection activities is proportional to the size of the class. Many other studies consider the cost of detecting faults in the class to be a fixed value and therefore evaluate models only by how well they detect faults. We believe that size of the class is a better cost indicator. At first we also considered a size of the change as possible effort estimation metric. It is, however, not enough to analyse only the modified code, since the modification can violate some more general class assumption and result in fault in a part of the class that was not modified. Therefore we selected size of the class for estimating analysis effort.

The *conclusion validity* concerns the correctness of conclusions we have made. When discussing conclusion validity we want to assess to what extent the conclusions we have made are believable. The conclusion validity is mostly interested in checking if there is a correct relationship (i.e., statistically significant) between the variables. Therefore, where possible, we have presented the statistical significance of our findings.

6. Conclusions

The goal of the study was to build prediction models that would increase the efficiency of fault detection in the modified code. We have built a number of models based on the data collected from a large telecommunication application. The models were later evaluated against three reference models: the model based on random selection of the classes for analysis, the theoretical best model, and a simple model based on the size of the class.

We have found that all our models provide an improvement compared to both random and size-based models. They are able to provide about 30% to 60% of the maximal theoretical improvement in fault detection efficiency.

We have found the models predicting the number of faults as most promising. They have proven to be more accurate and better compared to the fault-density prediction models.

A single metric that seems to have a large importance when it comes to fault prediction in modified code is the size of the modification (change size). This metric was included in all our models as well as selected the best single predicator for both the number of faults and the fault density.

In this study we have also managed to find empirical evidence for a number of popular hypothesis concerning faults. These findings concern the 60/20 rule (60% of the faults can be found in 20% of the code) as well as the fact that class size is a poor predicator of class fault-density.

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).

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] L. C. Briand, J. Wust, S. V. Ikonomovski, and L. H., "Investigating quality factors in object-oriented designs: an industrial case study", *Proceedings of the 1999 International Conference on Software Engineering*, 1999, pp. 345-354.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

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

[10] F. Fioravanti and P. Nesi, "A study on fault-proneness detection of object-oriented systems", *Fifth European Conference on Software Maintenance and Reengineering*, 2001, pp. 121-130.

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

[12] 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.

[13] M. Host, B. Regnell, and C. Wohlin, "Using Students as Subjects-A Comparative Study of Students and Professionals in Lead-Time Impact Assessment", *Empirical Software Engineering*, vol. 5, 2000, pp. 201-214.

[14] T. M. Khoshgoftaar, E. B. Allen, and J. Deng, "Controlling overfitting in software quality models: experiments with regression trees and classification", *Proceedings of the Seventh International Software Metrics Symposium*, 2000, pp. 190-198.

[15] T. M. Khoshgoftaar, E. B. Allen, and D. Jianyu, "Using regression trees to classify fault-prone software modules", *Reliability, IEEE Transactions on*, vol. 51, 2002, pp. 455-462.

[16] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Accuracy of software quality models over multiple releases", *Annals of Software Engineering*, vol. 9, 2000, pp. 103-116.

[17] T. M. Khoshgoftaar and N. Seliya, "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques", *Empirical Software Engineering*, vol. 8, 2003, pp. 255-283.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

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

[27] H. SungBack and K. Kapsu, "Identifying fault-prone function blocks using the neural networks - an empirical study", *Communications, Computers and Signal Processing, 1997. 10 Years PACRIM 1987-1997 - Networking the Pacific Rim. 1997 IEEE Pacific Rim Conference on*, vol. 2, 1997, pp. 790-793.

[28] C. Wohlin, *Experimentation in software engineering: an introduction*. Boston: Kluwer, 2000.

[29] 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.