Proceedings of the 13th Annual IEEE International Conference
on the Engineering of Computer Based Systems (ECBS 2006), Potsdam, Germany, 2006

The Accuracy of Fault Prediction in Modified Code – Statistical Model vs. Expert Estimation

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

Abstract

Fault prediction models still seem to be more popular in academia than in industry. In industry expert estimations of fault proneness are the most popular methods of deciding where to focus the fault detection efforts. In this paper we present a study in which we empirically evaluate the accuracy of fault prediction offered by statistical models as compared to expert estimations. The study is industry based. It involves a large telecommunication system and experts that were involved in the development of this system. Expert estimations are compared to simple prediction models built on another large system, also from the telecommunication domain. We show that the statistical methods clearly outperform the expert estimations. As the main reason for the superiority of the statistical models we see their ability to cope with large datasets, which results in their ability to perform reliable predictions for larger number of components in the system, as well as the ability to perform prediction at a more fine-grain level, e.g., at the class instead of at the component level.

1. Introduction

Large cost of finding and fixing faults in software has recently attracted a lot of attention. In literature we can find many case studies, according to which the activities connected with fault detection and removal account for a significant part of the project budget, e.g., in [6] 45% of the project resources were devoted to testing and simulation. Therefore, a lot of effort has been put into finding methods that attempt to increase the efficiency of fault detection.

One common approach is to build fault prediction models. The prediction models are particularly useful because of the Pareto principle [11]. It is a commonly know that in software systems faults are seldom distributed evenly. Usually small portions of code tend to contain a majority of faults, i.e., the Pareto principle (for a brief overview of research concerning the Pareto principle see [11]). By identifying the most fault prone modules prediction models make it possible to focus fault detection efforts (e.g., inspections) on the code that is most likely to contain faults, thus making fault detection more efficient. The fault prediction models usually aim at predicting either the number of faults [6, 7, 22, 29, 32] or the probability that the code unit contains faults [3, 4, 9, 13, 19].

Our experience shows that fault prediction models are much more popular in academia than in industry. Whether presented with the graphs describing the accuracy of fault prediction, statistical measures of model goodness, or percentages of correctly classified classes, the practitioners usually find it difficult to assess the practical value and accuracy of the model. The usual question is: "*Are your prediction models better than our guess?*". We find it difficult to give a good answer to that question. Neither our experience, nor the literature give us any indication what the truth might be. Therefore, we decided to set up a study comparing the accuracy of the fault prediction made by a fault prediction model vs. expert predictions.

In this paper we present the results obtained in this study. The study is industry based. We selected two large software projects done at Ericsson. One of them (System B) is used to build prediction models; the other one (System A) is used to evaluate them. To perform the expert estimation we invited six persons involved in the development of the System A. These were experienced designers and developers, who worked with the examined system for several years.

In this study we build prediction models for modified code. The reason is that when analyzing the code, we found that the most fault-prone code is the code introduced as modification to existing classes (for details see Section 3).

We assume that the cost of finding all faults in the class is proportional to the size of the class that contains these faults. Therefore, as the most efficient, we consider analyzing code units (e.g., classes, components) in descending fault density order.

The reminder of this paper is structured as follows: in Section 2 we present the work done by others in the area of expert estimations and fault prediction. In Section 3 we describe experts and systems from our study. Section 4 contains the description of methods we have used. In Section 5 we present the results. In Section 6 we discuss our findings. Section 7 contains the most important conclusions from this study.

2. Related Work

A lot of the research reports have been published in the area of improvement of fault detection. Mostly this research focuses on building fault prediction models. Depending on the output (the dependant variable), these fault prediction models belong to one of the following groups [20]: *Quality prediction models* quantify the quality, e.g. by predicting the number of faults in the code unit. Examples of such models can be found in [6, 7, 22, 29, 32]. *Classification models* classify code units as fault-prone or not. Examples of such models can be found in [4, 9, 13, 19].

Not only the models predict different things but very often they operate at different levels of the logical structure of the code. There are models that predict fault-proneness of classes [1, 5, 6, 9, 21, 32], modules [11, 18, 19, 25], components [23], or files [26].

The prediction is usually based on different characteristics of the examined code units. The most common candidates are different code metrics (e.g., [17, 28, 32]) or, for classes, variations of C&K [8] object oriented metrics (e.g., [4, 9, 32]). There are also studies that take historical information about code fault-proneness into account (e.g., [27, 28]).

Usually the prediction model construction starts with selecting independent variables (variables that are used to predict dependant variable). 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) [6, 10]. Therefore, some authors (e.g., [6]) build only simple models, i.e. containing one or at most two predicators.

Different methods are used in building prediction models. A commonly used method to select the best fault predicators is correlation analysis ([6, 9, 32]). The methods for building prediction models range from uni- and multivariate linear regression (e.g., [6, 7, 22, 25, 29, 32]) and logistic regression (e.g., [4, 9, 13, 19]) through regression trees (e.g., [17, 18]) to neural networks (e.g., [20, 31]).

Despite the fact that expert judgement is an accepted and practiced way of performing estimations [2, 16, 30] in many software related areas, we failed to find a lot of research that connects expert estimations with prediction of the fault-proneness of the individual code units. One example, was very interesting research [33, 34] in which expert estimations were used with statistical analysis as complementary methods. However, we failed to find any report presenting a comparative evaluation of expert judgments and statistical fault prediction models.

3. Study Subjects

In this study we use two software systems (System A, and System B) developed by Ericsson. Both systems are working within the service layer of mobile phone network. We use System B to build prediction models that are later evaluated by applying them to System A. Because System A is used for evaluating, all experts invited to participate in this study are involved in the development of System A.

3.1 Systems Under Study

We use the most current releases of System A and System B. These are large telecommunication systems. Their sizes are about 800 classes (500 KLOC), and 1000 classes (600 KLOC) for System A and System B, respectively. Both systems are mature and have been on the market for over 6 years. Both systems are implemented using C++. One of the projects has been developed in Sweden. The other one has mostly been developed in China.

When analyzing the code we found that the most fault-prone code is the code introduced as modification to existing classes. In System A the modification of classes from the previous release accounted for 37% of the code written in the current release (63% of the new code was introduced as new classes). These 37% of the code contained 62% of the faults found in the project release we examine in this study. A similar trend has also been observed in System B - 44% of the code was introduced as modification to classes from the previous release. These 44% of the code contained 78% of all faults. Therefore, in our study we decided to focus specifically on the modified code.

Both systems are divided into components, which contain classes. The number of components modified in the examined releases of the products was 35 in System A, and 41 in System B. That corresponds to 249 modified classes in System A and 319 modified classes in System B. The information about faults was available both at the class and component level.

3.2 Participating Experts

We invited six experts to participate in the study. All of them are directly involved in the development of System A. All of them have several years of working

experience with System A. Their major tasks within System A are design and implementation.

At the time of the study, the implementation of the examined release of System A was finished. All the experts are familiar with the architecture of System A. They all know what kind of functionality is implemented in each component. They also have extensive knowledge and experience in the system domain – telecommunications. All the experts know the scope of the current release of System A– they know what functionality was implemented and where.

One risk of such a study set-up is that the developers may basically know the fault distribution. We believe it has not been the case in our study. The work in the project is organized according to the "component responsibility" idea - each developer is fully responsible for one or more components. Because of that, in practice, the developers do not have a global picture concerning fault distribution – they only receive information concerning the faults that were found in the components they are responsible for. Normally, they are not provided with any global statistics concerning faults. Therefore, their prediction concerning the faults made in this study is not based on any global statistics but on their own "gut-feeling", based on experience and knowledge of the project.

4. Methods

In this section we present the methods we use in this study. Section 4.1 presents methods we used to build prediction models. Section 4.2 presents the way we collected and used the data gathered from our experts. In Section 4.3 we present the way in which we evaluated and compared both methods.

4.1 Building Prediction Models

Our models predict the fault density of individual code units. This can be predicted in two ways:

- by predicting fault density (Faults/Size) – 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 size (Size) of the code unit – Faults are predicted by the model, while size (Size) is measured.

We collected the data for prediction at the class and at the component level. The class level metrics are summarized in Table 1. These are mostly C&K [8] design metrics, and code metrics. The component level metrics are summarized in Table 2. These are only code metrics. In the components we performed measurements only on classes that were modified.

Similarly to [6], we build simple prediction models based on one predicator only. Such models do not suffer from multicolinearity risk [10]. Therefore, simple models are more likely to be stable over releases. Additionally, such models require less data to be collected compared to multivariate models. Obviously, by using one metric only, we deliberately give up the potential benefit from introducing more information, carried by other metrics, into the model.

To select the best single fault predicators we performed a correlation analysis. A correlation analysis is commonly used for that purpose by other researchers [24, 32]. It quantifies the relation between two metrics as a value between -1 and 1. An absolute value close to one characterizes good predicator variables. Values close to zero indicate very weak linear relationship between the variables and thus low applicability of one variable to predict the other.

Table 1. Metrics collected at the class level.

Name	Variable	Description
Independent metrics		
Coup	Coupling	Number of classes the class is coupled to [8, 12]
NoC	Number of Children	Number of immediate subclasses [8]
WMC	Weighted Methods per Class	Number of methods defined locally in the class [8]
RFC	Response for Class	Number of methods in the class including inherited ones[8]
DIT	Depth of Inheritance Tree	Maximal depth of the class in the inheritance tree[8, 11]
LCOM	Lack of Cohesion	*"how closely the local methods are related to the local instance variables in the class"* [12]. In the study LCOM was calculated as suggested by Graham [14, 15]
Stmt	Number of statements	Number of 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 variables		
Faults	Number of faults	Number of faults found in the class
FaultDensity	Fault density	Fault density of the class

Table 2. Metrics collected at the component level.

Name	Variable	Description
Independent metrics		
CompStmt	Number of statements	Number of statements in the component (only statements from modified classes in the component were counted)
CompMeth	Number of methods	Number of statements in the component (only methods from modified classes in the component were counted)
CompClass	Number of modified classes	Number of modified classes in the component
CompChg	Change size	Number of new and modified LOC (compared to previous release)
Dependent variables		
CompFaults	Number of faults	Number of faults found in the component
CompFaultDensity	Fault density	Fault density of the component

To build our prediction models we use the univariate linear regression. The univariate linear regression estimates value of the dependant variable (number of faults or fault-density) as a function of the independent variable [24]:

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

4.2 Expert Estimation

The expected outcome of the expert estimation was a ranking of the code units according to their decreasing fault density. In the beginning of the study we performed a number of interviews to establish an appropriate level to perform the expert predictions. The question was if the experts should perform estimations at the class or at the component level. It quickly turned out that the class level presents too fine-grained information. Even though the experts knew what each component did, it was very difficult for them to predict the responsibility of particular classes. Additionally, the amount of data (249 classes) was considered unmanageable. The number of components (35 in System A) was much smaller. Therefore, the expert estimation was performed only at the component level.

The expert estimation was done individually by each of our experts. Later the individual rankings were used as an input to a consensus meeting. The goal of the meeting was a joint ranking of the components.

During the individual rankings the experts were provided with a list of modified components. Additionally, for each component, we enclosed information concerning the subsystem the component belonged to, as well as the accumulated size of modified classes within the component, which is considered the cost of analyzing the component in our study. The experts were asked to indicate in which order they would analyze the components, so that they analyze the components with the highest fault densities first. Each expert was given an explanation concerning our study in order to assure the full understanding of the task they were asked to perform.

The experts were allowed not to rank all the components. In fact, it turned out that no one ranked more than eight components. The experts said that they are able to identify a couple of the most fault prone components, but after a certain point they would put the components in random order.

4.3 Prediction Evaluation

We evaluate the statistical prediction models and the expert predictions from the perspective of the increase of the efficiency of fault detection they provide. Their goodness is measured by the amount of code necessary to analyze to detect a certain number of faults, i.e., one prediction is better if we are able to detect more faults by analyzing the same amount of code compared to another prediction. Therefore, we perform the evaluation by plotting the percentage of faults that would be detected if analyzing System A according to either the experts' suggestions or the prediction model against the accumulated percentage of code that would have to be analyzed.

In order to obtain a point of reference for our evaluations, we introduce two reference models:

- *Random model* – the model describing completely random search for faults
- *Best model* – the theoretical model in which code units are ordered by their actual fault density

The Random model provides a baseline for our models as it describes what results, on average, we could expect if we analyzed the code not following any model. We believe that we should evaluate the predictions against the Random model because only in this way we are able to assess their practical value.

The Best model provides a boundary of how good the prediction can be. Obviously, the Best model would be different for predictions at the class and at the component level. The class level prediction has finer granularity and therefore, at least theoretically, is able to provide more precise results. By comparing the Best model made for components and for classes we can assess the practical value of having finer granularity prediction.

The evaluation of model predictions vs. expert estimations is performed by checking how each

Table 3. Correlation analysis results at the class and the component level. The highest correlations for respective levels (class, component) are marked in bold.

	Class level metrics									Component level metrics			
	Coup	NOC	WMC	RFC	Stmt	MaxCyc	DIT	LCOM	ChgSize	Comp Chg	Comp Stmt	Comp Meth	Comp Class
Faults	0.25	-0.01	0.14	0.04	0.26	0.31	-0.07	0.13	**0.60**	**0.79**	0.63	0.35	0.55
Fault Density	0.06	-0.01	-0.01	-0.03	-0.02	0.01	-0.01	0.05	0.20	0.21	0.07	0.01	0.09

particular solution performs compared to the Best model, the Random model, and to each other. The closer the prediction is to the Best model the better it is. If the prediction is better than the Random model then we can say that using it presents an improvement over not using any method at all.

5. Results

5.1 Building Prediction Model

As described in Section 4.1, the building of prediction models begins with a correlation analysis. The goal of the correlation analysis is to select the best individual fault predicator. The correlation analysis was performed for both class and component level metrics. The results of correlation analysis are presented in Table 3.

As it can be noticed in Table 3, the most promising fault predicator for both classes and components is the size of the modification (ChgSize metric at class level, and CompChg metric at the component level). The highest correlations are marked in bold in Table 3. In both cases the correlation coefficients are the highest when predicting the number of faults. Therefore, we build models that predict the number of faults and we divide their output by the size of the respective code unit, i.e., class or component.

The models based on ChgSize and CompChg are built using linear regression. The results are presented in Table 4. As both models are based on the information concerning the size of the modification, not surprisingly the models are quite similar.

5.2 Expert Estimation

The distribution of "votes" of experts, the group consensus, and the actual ranks of the components are presented in Table 5. Only the components that were selected by at least one expert are presented.

In the individual rankings neither of our experts ranked all 35 components. Each expert ranked between 5 and 8 components. Altogether experts pointed out 15 different components, i.e., neither had any opinion about the fault-proneness of the remaining 20 components. These 15 ranked components account for about 60% of the code.

The "group consensus" was apparently more difficult to reach than individual rankings because the experts ranked only 4 components i.e., less than in any individual ranking. These four components accounted for about 30% of the code.

In Table 5 it can be noticed that in the individual rankings the components can be divided into two subgroups. One subgroup contains components that were selected by a majority of the experts, i.e., four and more experts pin-pointed them. These are components with numbers: 1, 3, 4, 5, 9, 10. To the other group belong the components selected only by one or two interviewees, i.e., components with numbers: 2, 6, 7, 8, 11-15. It is quite clear that apart from two exceptions (Component 1, and Component 10) there is a rather large discrepancy between the ranks assigned by experts to the components. This means that, despite the fact that most experts considered a certain component fault-prone, their estimation of its fault-density was different.

All components ranked in the "group consensus" ranking are the components that were selected by the majority of experts in the individual rankings. Component 1 and Component 10 were ranked according to the trend from the individual rankings, which is not surprising - the experts were consistent in ranking them as the first and the third in their individual rankings. Ranking Component 9 and Component 4 as the second and the fourth respectively must have been an outcome of the group discussion,

Table 4. Prediction models build in the study. "Prediction level" indicates if the models works at class or at component level. "Model calculated" is the model obtained by linear regression. "Model applied" is the transformation of the "Model calculated" so that it predicts fault density instead of the number of faults.

Model name	Prediction level	Model calculated	Model applied
ComponentPred	Component	Faults = 0.002*ComChg+0.209	FaultDensity = (0.002*ComChg+0.209) / CompStmt
ClassPred	Class	Faults = 0.002*ChgSize+0.018	FaultDensity = (0.002*ChgSize+0.018) / Stmt

Table 5. The rankings of individual experts and the joint ranking of all experts. Only 15 components out of 35 were selected, and only those components are presented in the table below. Lower rank value indicates higher fault-density in the component predicted by expert. "Correct ranking" is the actual rank of the component when all 35 components are ranked.

	Component														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Expert1 ranking	2		4	1					5		3				
Expert2 ranking	1				2			4		3			5		
Expert3 ranking	1	5	7	6	4				2	3					
Expert4 ranking	1		3	6	7		5		4	2					
Expert5 ranking	2		1		7					3	6	5	8		4
Expert6 ranking			2	1	8	7		6	5	4				3	
Group consensus ranking	1			4					2	3					
Correct ranking	10	5	9	8	22	18	12	25	17	6	16	7	26	24	13

because such ranking was not suggested by any individual expert.

At this point we can also perform some partial assessment of expert prediction accuracy. By comparing the results of individual experts with the actual ranking of components (the last row in the table) we can see that out of the 15 components they pointed to, only 8 belong to the actual 15 components with the highest fault densities. By comparing the "consensus group" ranking with the actual ranking of components we can see that neither of the components pin-pointed by the experts belongs to the top 4 components with highest fault-density.

5.3 Prediction evaluation

Our prediction starts with reference models building. We build three reference models, one describing an average result of random picking of code units for analysis (Random model), and two models describing the theoretical best result that can be obtained. One of them describes the maximum that can be obtained when predicting faults at the class level (Best model class), the other when predicting at the component level (Best model component). These models are presented in Figure 1.

As it can be noticed in Figure 1 the Random model is quite far from the best possible model. For example, by analyzing 20% of the code randomly we can find 20% of faults. By analyzing appropriate 20% of code we should find up to 70% of the faults in case of class level prediction and up tp 50% of faults if the prediction was made at the component level. This also indicates that, at least theoretically, the class level prediction should give better results, because the class level prediction is more fine-grained and therefore more precise. From Figure 1 we see that theoretically

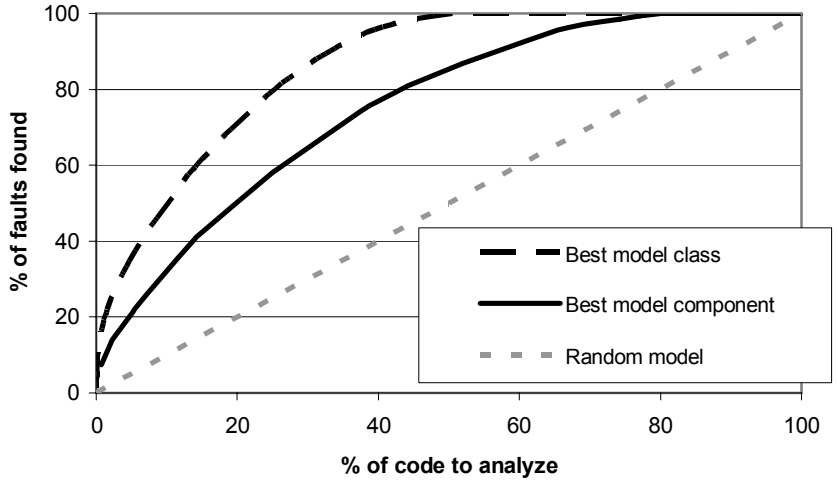

Figure 1. Reference models - evaluation

the component level prediction can provide about one-third of the improvement over the random model offered by the class level prediction (in Figure 1 the distance between Best model component and Best model class is more or less equal to 1/3 of the distance between the Best model class and Random model).

The evaluation of prediction models and expert estimations is presented in Figure 2. We present all the individual expert estimations, "group consensus", both statistical prediction models (ClassPred, ComponentPred) and all the reference models.

From Figure 2 we can conclude that both statistical prediction models clearly outperform expert estimations. They not only offer higher accuracy in the range of code covered by expert estimations (approximately up to 50% of code) but also provide prediction, which is better compared to the Random model for the rest of the code. By comparing ClassPred with the best of expert estimations for the percentage of code covered by expert estimations we can see that ClassPred offers about three times as big improvement over the Random model as the best of expert estimations.

Other findings from Figure 2 concern the gain from using more fine grained information and predicting at the class level. As we can see, there is a clear practical gain connected with predicting at the class level. For example, for the range of code covered by expert estimations the gain from using ClassPred is almost equal to the maximum possible gain from using any component level prediction model, i.e., the Best model component. Our component level prediction model ComponentPred is not even close to that.

Surprisingly for us, the "Group consensus" estimation turned out to be one of the worst estimations made by the experts. Some of individual estimations were actually not only more correct but they also accounted for more code.

6. Discussion

6.1 Findings

The results obtained in our study seem to support the idea of building fault prediction models. By comparing the accuracy of fault prediction, and the increase of fault detection efficiency connected with it, we showed that statistical models have two major advantages over human estimation.

The first advantage is that they are not affected by the size of dataset and therefore they are able to rank the entire project. The experts that we were invited to participate in this study were able to rank 5 to 8 out of 35 components. The second advantage of prediction models is that they can easily operate on fine-grained data. In our case, while the class level prediction was not feasible for our experts to perform, the statistical prediction model accomplished this task with very

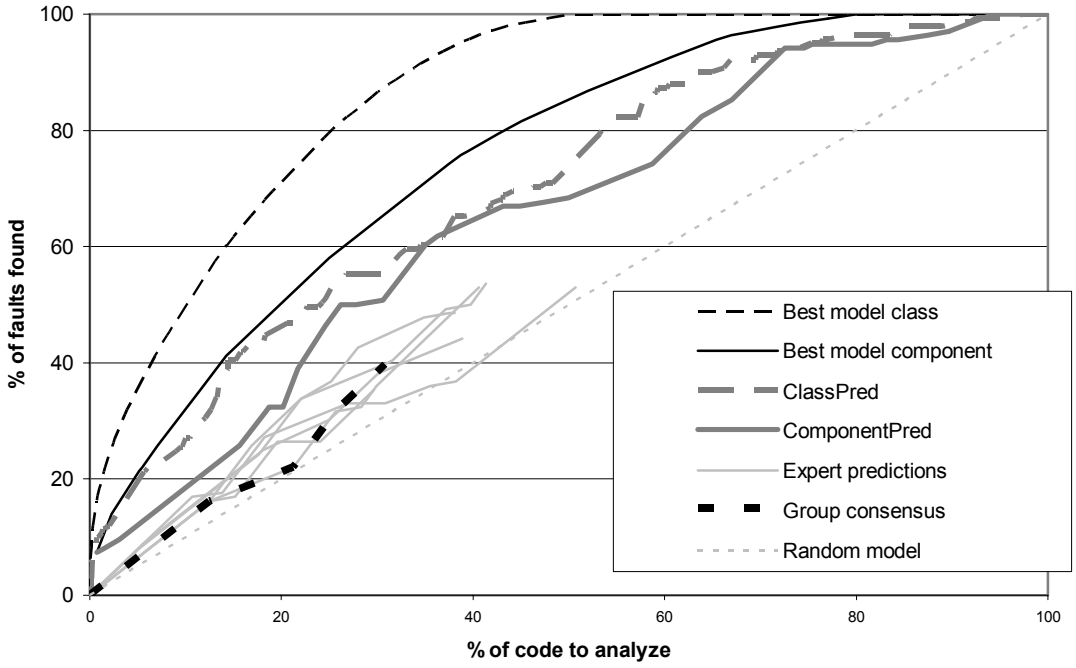

Figure 2. Prediction model vs. expert estimation – evaluation.

good results. This advantage is partially caused by the statistical model's ability of coping with large datasets, but also by human inability to grasp and understand large system structures (e.g., 250 classes in our case).

There is also one more advantage of predication models, not evaluated in this study. It is their cost. They are reasonably cheap to build and even cheaper to apply – normally they can be implemented in the form of e.g., script that collects and processes all the information automatically. Expert estimation is more expensive, since for each project it must be set up and performed independently. Obviously to perform expert estimation we need experts – sometimes, in relatively new projects, or when projects are overtaken by other team, the experts might simply not be available.

On the other hand an expert estimation has other advantages. A statistical prediction model predicts faults, but it does not classify them in any way. Naturally, not all faults are the same – some of them might be more difficult to find than the others, because they may be related to the complexity of the functionality that the code implements. Some faults may be more severe than the others. It is possible that the experts tend to pin-point more correctly the components that are more likely to contain this kind of faults. Due to the lack of appropriate data concerning fault severity we could not verify this hypothesis in our study. Also the experts take into account information that is not present in the statistical model – e.g., in our case the size of the change is considered to be the best fault predicator. However, in the project, it might happen so, that even though the change in the component is relatively small, there was a number of people involved in introducing it, which may make it more prone to faults compared to the change introduced by a single designer.

The results obtained in our study quite clearly show that prediction models outperformed expert estimation in almost every aspect. They were reasonably accurate and consistent in providing improvement over the Random model for any percentage of the code. Also, when analysing expert estimations, we can find a number of worrying factors. The experts did not agree with each other, they either selected different components, or, if they selected the same components, they estimated their fault density differently. From Table 3 we can see that even though most of the experts agreed on high fault densities in some of the components, in fact the components they selected are actually not the most fault-prone in the entire system. Neither of the experts mentioned any of the actual four most fault-prone components in the system.

6.2 Validity

The reader must bear in mind that this paper has been meant more as an experience report than a formal experiment. Our selection of projects, and selection of experts are convenience-based – we selected projects and experts that were available to us. Probably selecting different experts could have given different results. Also the entire exercise was not performed as a controlled experiment and we can not assure that e.g., the experts did not have some partial knowledge about the actual fault-proneness of the components. The number of the experts was also small and therefore it is difficult to generalize their performance to the performance of experts at all.

However, we still believe that some general lessons can be learned from our study. It seems quite probable that most experts would face the problems our experts faced, e.g., problems with coping with large amounts of fine-grained data. It is also sure that prediction on a low level, like e.g., the class level, would give better results compared to prediction on a higher level, e.g., the component level. We are almost sure that for most medium-to-large system the class level prediction is not feasible to be performed by people.

We also believe that the statistical prediction models obtained in our study are general. When building them we followed a good academic practise of building models on other system than the system used to evaluate the models.

Most issues concerning expert estimation validity, like the experts' possible knowledge about actual fault distribution, should result in better than average performance of experts. This might be considered an argument supporting our conclusions, because, even with this "handicap", the expert estimations where outperformed by the statistical prediction models.

7. Conclusions

The goal of this study was to compare the accuracy of fault prediction made by a statistical fault prediction model and by human experts. To perform the comparison we built two simple statistical fault prediction models and we asked a number of experts to perform the individual as well as group estimations of fault-proneness of components from a large telecommunication system.

The evaluation was performed from the perspective of fault detection efficiency increase that could be obtained if analyzing the code according to suggestions made by experts or made by the statistical models. Additionally, the models were evaluated against three reference models: model based on a random selection of the code units for analysis, the theoretical best model for predicting faults at the class level, and the

theoretical best model for predicting faults at the component level.

We found that both expert estimations and statistical models actually provide an improvement over the random search for faults. When it comes to comparing both methods we found that statistical models outperform expert estimations. For the percentage of code covered by the expert estimations the best statistical fault prediction model offered three times as big improvement over the random search for faults as compared to the best of expert estimations. Moreover, it continued to provide an improvement even after the point where the experts gave up.

We identified two reasons for statistical models being better. Statistical models are not affected by the size of dataset, while human ability to grasp and assess the complexity of large systems is seriously limited. The ability to deal with large datasets gives the statistical models an ability to perform more fine-grained estimations, i.e., estimations at lower level.

In this study we showed that the more fine-grained prediction (e.g., prediction at the class instead of at the component level) is not only better from theoretical but also from practical perspective. Our class level prediction model was more accurate compared to the component level prediction model. In fact, it was very close to the theoretical maximum the component level prediction can provide.

In the study we also discussed other advantages and disadvantages of statistical prediction models and expert estimations. Apart from what we have mentioned before, the statistical methods have also two other important advantages – there are reasonably cheap to build and apply, as well as they can be used in the absence of experts (e.g., when project is transferred to another development organization). On the other hand expert estimations can take into account issues not present in the statistical model, e.g., severity of the faults, testability of the components, or basically factors other than those that are taken into account in the prediction model.

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] V. R. Basili and L. C. Briand, "A validation of object-oriented design metrics as quality indicators." *IEEE Transactions on Software Engineering*, vol. 22, 1996, pp. 751-762.

[2] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

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

[4] 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.

[5] L. C. Briand, J. Wust, S. V. Ikonomovski, and L. H., "Investigating quality factors in object-oriented designs: an industrial case study", *Proc. of the 1999 Int'l Conf. on Software Eng.*, 1999, pp. 345-354.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] R. T. Hughes, "Expert judgement as an estimating method", *Information and Software Technology*, vol. 38, 1996, pp. 67-76.

[17] T. M. Khoshgoftaar, E. B. Allen, and J. Deng, "Controlling overfitting in software quality models: experiments with regression trees and classification", *Proc. of The 17th International Software Metrics Symposium*, 2000, pp. 190-198.

[18] 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.

[19] 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.

[20] 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.

[21] X. Li, Z. Liu, B. Pan, and D. Xing, "A measurement tool for object oriented software and measurement experiments with it" presented at 10th International Workshop New Approaches in Software Measurement, Berlin, Germany, 2001.

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

[23] M. C. Ohlsson, A. Andrews Amschler, and C. Wohlin, "Modelling fault-proneness statistically over a sequence of releases: a case study", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, 2001, pp. 167-199.

[24] 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.

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

[26] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the Location and Number of Faults in Large Software Systems", *IEEE Transactions on Software Engineering*, vol. 31, 2005, pp. 340-355.

[27] 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.

[28] 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.

[29] 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.

[30] M. Shepperd and M. Cartwright, "Predicting with sparse data", *IEEE Transactions on Software Engineering*, vol. 27, 2001, pp. 987-998.

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

[32] 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.

[33] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques", *IEEE Intelligent Systems*, vol. 19, 2004, pp. 20-27.

[34] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation", *Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering*, 2004, pp. 149-155.