Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 80 (2007) 1227–1238
The Journal of Systems and Software
www.elsevier.com/locate/jss

Statistical models vs. expert estimation for fault prediction in modified code – an industrial case study

Piotr Tomaszewski *, Jim Håkansson, Håkan Grahn, Lars Lundberg

Department of Systems and Software Engineering, School of Engineering, Blekinge Institute of Technology, SE-372 25 Ronneby, Sweden

Received 2 December 2005; received in revised form 12 December 2006; accepted 19 December 2006
Available online 22 December 2006

Abstract

Statistical fault prediction models and expert estimations are two popular methods for deciding where to focus the fault detection efforts when the fault detection budget is limited. In this paper, we present a study in which we empirically compare the accuracy of fault prediction offered by statistical prediction models with the accuracy of expert estimations. The study is performed in an industrial setting. We invited eleven experts that are involved in the development of two large telecommunication systems. Our statistical prediction models are built on historical data describing one release of one of those systems. We compare the performance of these statistical fault prediction models with the performance of our experts when predicting faults in the latest releases of both systems. 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. This makes it possible for statistical models to perform reliable predictions for all components in the system. This also enables prediction at a more fine-grain level, e.g., at the class instead of at the component level. We show that such a prediction is better both from the theoretical and from the practical perspective.

Keywords: Fault prediction; Expert estimation; Evaluation

1. Introduction

The high cost of finding and correcting faults in software projects has become one of the major cost drivers of software development. In literature, we can find many case studies, which show that the activities connected with fault detection account for a significant part of the project budget. On the other hand, software projects often face budget limitations that put stringent restrictions on extensive and expensive quality assurance. To achieve the highest possible product quality, software developers need to decide where to focus their fault detection efforts in order to detect as many faults as possible within a given budget.

If we assume that the cost of fault detection (e.g., inspection) for a code unit (e.g., a class or a component) is related to the size of this unit, we can see that fault detection is most efficient when it is focused on the code units with the highest *fault density*, i.e., with the largest number of faults per line of code. This assumption implies that if our fault detection budget is limited and we can not cover all the code in the system then ideally we should perform our fault detection activities (e.g., inspections) on code units in the order of their decreasing fault density. This would guarantee that we detected as many faults as it was possible within given budget.

It is commonly known that faults are very rarely distributed evenly in software systems. Typically, the majority of faults can be found in a minority of the code units (for an overview of research concerning this issue see Fenton and Ohlsson, 2000). Therefore, when the budget is limited, fault detection should be performed on the code units in order of their decreasing fault density. To plan such fault detection, we must be able to predict the fault density of the code units.

* Corresponding author. Tel.: +46 457 385876; fax: +46 457 27125.
E-mail addresses: piotr.tomaszewski@bth.se (P. Tomaszewski), hakan.grahn@bth.se (H. Grahn), lars.lundberg@bth.se (L. Lundberg).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.12.548

One approach to perform fault density prediction is to build statistical fault prediction models. Such models attempt to predict fault-proneness of the code units based on their characteristics, e.g., size, complexity, etc. This approach is very popular in academia – there is a lot of research describing and evaluating such models (Briand et al., 2000; Cartwright and Shepperd, 2000; Chidamber et al., 1998; El Emam et al., 2001; Fioravanti and Nesi, 2001; Khoshgoftaar et al., 2000a,b; Nikora and Munson, 2003; Ping et al., 2002; Zhao et al., 1998). Our own experience shows, however, that fault prediction models are less popular and not so widespread in industry.

Another approach to predict the fault-proneness of the code units are expert estimations. In this approach, human experts suggest the order in which the code units should be analyzed. The experts usually base such decisions on their experience and knowledge about the system. In contrary to the statistical prediction models, this approach seems to be very popular in industry but is not well researched (for an interesting discussion why expert estimations are popular in industry as well as an overview of studies related to performing expert estimations in software projects see Hughes, 1996).

Even though an expert judgement is an accepted and a common way of performing estimations in many software engineering related areas (Boehm, 1981; Hughes, 1996; Shepperd and Cartwright, 2001) we have failed to find any report presenting a comparative evaluation of the applicability of expert judgments vs. statistical prediction models for predicting the fault-proneness of individual code units. Therefore, in this study our goal is to compare the accuracy of the fault prediction made by statistical fault prediction models with the accuracy of expert estimations.

The study is industry based. As study objects we have selected two large software systems from the telecommunication domain developed at Ericsson. We denote them as System A and System B. We use one release of System A and two releases of System B (called System B1 and System B2). System B1 is used to build our statistical prediction models. The models are evaluated on System A and System B2. To perform the expert estimation we have invited six persons involved in the development of System A and five persons involved in the development of System B2.

Each system release that we examine in this study introduces a significant amount of new functionality. Typically, the new functionality is introduced either as new classes or as modifications of existing classes. In our dataset we have found that code inserted as modifications of existing classes accounts for a minority of the code introduced in each system's release. At the same time the modified classes contained a majority of the faults (see Section 3.1 for details). Therefore, in this study we focus specifically on predicting fault density in the modified code.

The reminder of this paper is structured as follows. In Section 2, we present the work done by others in the area of fault prediction. Section 3 contains more detailed information concerning the systems and the experts in our study. In Section 4, we introduce the methods that we use in this study. In Section 5, we present the results, and in Section 6, we discuss our findings and their validity. Section 7 contains the most important conclusions from our study.

2. Related work

A lot of work has been done in the area of fault detection improvement. A large portion of 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 (Khoshgoftaar and Seliya, 2003):

- Quality prediction models – these models attempt to quantify the quality of the code unit, e.g. by predicting the number of faults in the code unit. Examples of such models can be found in Cartwright and Shepperd (2000); Chidamber et al. (1998); Nikora and Munson (2003); Ping et al. (2002); Zhao et al. (1998).
- Classification models – these models classify code units as fault-prone or not, i.e., they predict if the code unit contains faults. Examples of such models can be found in Briand et al. (2000); El Emam et al. (2001); Fioravanti and Nesi (2001); Khoshgoftaar et al. (2000a,b).

The models often operate at different levels of the logical structure of the code. There are models that predict fault-proneness of classes (Basili and Briand, 1996; Briand et al., 1999; Cartwright and Shepperd, 2000; El Emam et al., 2001; Li et al., 2001; Zhao et al., 1998), modules (Fenton and Ohlsson, 2000; Khoshgoftaar et al., 2002; Khoshgoftaar et al., 2000a,b; Ohlsson et al., 1998), components (Ohlsson et al., 2001), or files (Ostrand et al., 2005).

The prediction models are usually based on different characteristics of the code units. These characteristics are commonly presented in the form of different code metrics (e.g., Khoshgoftaar et al., 2000a,b; Pighin and Marzona, 2005; Zhao et al., 1998) or, for classes, variations of C&K Chidamber and Kemerer (1994) object oriented metrics (e.g., Briand et al., 2000; El Emam et al., 2001; Zhao et al., 1998). There are also studies that take historical information about fault-proneness of code units into account (e.g., Pighin and Marzona, 2003; Pighin and Marzona, 2005).

The construction of a prediction model usually starts with the selection of the independent variables (i.e., the variables that are used to predict a 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 a lower practical value (more metrics have to be collected) (Cartwright and Shepperd, 2000; Fenton and Neil, 1999). Therefore, some authors (e.g., Cartwright and Shepperd, 2000) focus on

building only simple models, containing one or at most two predicators (independent variables).

A commonly used method to select the best fault predicators is correlation analysis (Cartwright and Shepperd, 2000; El Emam et al., 2001; Zhao et al., 1998). The methods for building prediction models range from uni- and multivariate linear regression (e.g., Cartwright and Shepperd, 2000; Chidamber et al., 1998; Nikora and Munson, 2003; Ohlsson et al., 1998; Ping et al., 2002; Zhao et al., 1998) and logistic regression (e.g., Briand et al., 2000; El Emam et al., 2001; Fioravanti and Nesi, 2001; Khoshgoftaar et al., 2000a,b) through regression trees (e.g., Khoshgoftaar et al., 2000a,b; Khoshgoftaar et al., 2002) to neural networks (e.g., Khoshgoftaar and Seliya, 2003; SungBack and Kapsu, 1997).

Despite the fact that expert judgements are an accepted and widely practiced way of performing estimations (Boehm, 1981; Hughes, 1996; Shepperd and Cartwright, 2001), we have found only very little research that connects subjective expert estimations with statistical predictions based on the characteristics of individual code units. The examples that we have found are studies (Zhong et al., 2004a,b) in which expert estimations are used together with statistical analysis as complementary methods – statistical methods are used to group code units with similar characteristics and then, it is up to an expert to estimate if a given group of code units is fault-prone. We have, however, failed to find any report presenting a comparative evaluation of expert judgments and statistical fault prediction models.

Subjective expert judgements were, however, evaluated in the context of inspection effectiveness. Inspections effectiveness is approximated by estimating the number of faults that remain in a software artefact (document, code unit) after the inspection is completed (Biffl et al., 2000; El Emam et al., 2000; Thelin, 2004). This estimation can later be used to decide where to direct the additional quality assurance efforts (i.e., which artefacts to re-inspect) and, therefore, these studies are somewhat similar to our study. Some positive results were reported when using subjective judgements for estimating inspection effectiveness (Biffl et al., 2000; El Emam et al., 2000). However, an empirical evaluation presented in Thelin (2004) showed that objective methods (e.g., capture-recapture) outperformed such subjective judgements. However, as we see it, it is not entirely clear if the conclusions from these studies can apply to the case that is presented in this paper. The experts in those studies were predicting a different thing, i.e., the actual number of faults. In those studies, in contrast to our study, the estimations were made by experts after they performed an initial inspection of an artefact. Also, the studies presented above were experimental and for that reason the sizes of systems under study were very small as compared to our study (our results indicate that in case of such studies "size matters" – see Section 6.1). Also in Biffl et al. (2000) and Thelin (2004) the estimations concerned the number of faults in the documentation, not in the code.

3. Study objects

3.1. Systems under study

In this study we use the most current release of System A and two latest releases of System B. These are large telecommunication systems. The sizes of these systems are about 800 classes (500 KLOC), and over 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. Over that time the systems have evolved – a number of releases of each of them have been produced. Both systems are implemented in object oriented technology using the same programming language. One of the systems has been developed in Sweden. The other one has mostly been developed in China and is currently being transferred to Sweden.

The systems are logically divided into a number of subsystems. Each subsystem is built of components. Each component consists of a number of classes. The numbers of components that have been modified in the examined releases of the products are 35 in System A, 41 in System B1, and 43 in System B2. That corresponds to 249 modified classes in System A, 319 modified classes in System B1, and 180 modified classes in System B2. The information about faults is available at the class level. Therefore, we are able to assign faults to the particular classes and, through them, to the components.

When analyzing the code and the fault data we have found that in all three cases (System A, System B1, and System B2) the most fault-prone code is the code introduced as modifications of existing classes. In System A, the code introduced as modifications of the classes from the previous release accounts for 37% of the code written (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 that we examine in this study. A similar trend has also been observed in System B. In System B1, about 44% of the introduced code modifies classes from the previous release. These 44% contain 78% of all faults. In System B2, the modified code accounted for 45% of code introduced in this release. The modified classes in System B2 contained 59% of faults. It seems that in the systems under study the modified code units were more fault-prone than new ones. Therefore, in this study we focused on predicting fault-densities only in modified code units.

3.2. Participating experts

In total, we have invited eleven experts to this study. Six of them have been involved in the development of System A, and five of them have been involved in the development of System B2. The experts were appointed by members of the respective software development projects as the persons most qualified to perform such predictions. All of our experts have several years of working experience with telecommunication systems. Their main tasks are system design and implementation. All our experts are familiar

with the architectures and functionalities of their respective systems. They also know the scopes of the releases under study. They know what functionality was added in the releases they were asked to perform estimations for.

The major difference between experts involved in performing estimations concerning System A and those performing estimations concerning System B2 is their experience with the respective products. System B2 is currently being transferred from an offshore development site. Therefore, all of the experts involved in performing estimations concerning System B2 have limited experience (up to one year) of working with System B2, as compared to the six-year experience of the experts involved in the development of System A.

At the time of the study, the development of the examined releases of the systems was finished. One risk of such study set-up is that the experts may basically know the fault distribution. We believe it has not been the case in our study. First, such statistics are actually never made public in a project. Second, the work in the project is organized according to the "component responsibility" principle – each developer is fully responsible for one or more components. Because of that, in practice, when faults are discovered and reported the developers do not have a global picture concerning the fault distribution – they are only interested in information concerning the faults that were found in the components they are responsible for at a given point of time. Therefore, their predictions concerning the faults made in this study are not based on any global statistics but on their own "gut-feeling", based on experience and knowledge of a system under study and the scope of a project.

4. Methods

In this section, we present the methods, which we use in this study. Section 4.1 presents the methods we used to build our 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 methods we use to evaluate and compare our prediction models with the estimations of our experts.

4.1. Building prediction models

Our prediction models are built based on data from System B1. The goal of our models is to predict fault density of different code units. As we see it, the fault density can be predicted in two ways:

- By predicting the 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 is measured.

In this study, we have collected data that makes it possible for us to perform predictions both at the class and at the component level. The metrics collected at the class level are summarized in Table 1. These are mostly C&K (Chidamber and Kemerer, 1994) design metrics, and code metrics. The metrics collected at the component level are summarized in Table 2. These are simple code metrics measuring the size of the component and the size of the change. Within the components we have performed measurements only on those classes that were modified.

Similarly to (Cartwright and Shepperd, 2000), we have decided to build simple prediction models that are based on one predicator only. Such models do not suffer from the risk of multicolinearity, which is a typical risk for multivariate models (Fenton and Neil, 1999). Therefore, simple models are usually more likely to be stable over releases.

Table 1
Metrics collected at the class level

Name	Variable	Description
Independent metrics		
Coup	Coupling	Number of classes the class is coupled to (Chidamber and Kemerer, 1994; Fenton and Pfleeger, 1997)
NoC	Number of children	Number of immediate subclasses (Chidamber and Kemerer, 1994)
WMC	Weighted methods per class	Number of methods defined locally in the class (Chidamber and Kemerer, 1994)
RFC	Response for class	Number of methods in the class including inherited ones (Chidamber and Kemerer, 1994)
DIT	Depth of inheritance tree	Maximal depth of the class in the inheritance tree (Chidamber and Kemerer, 1994; Fenton and Ohlsson, 2000)
LCOM	Lack of cohesion	"How closely the local methods are related to the local instance variables in the class?" (Fenton and Pfleeger, 1997). In the study, LCOM was calculated as suggested by Graham (Cartwright and Shepperd, 2000; Chidamber et al., 1998; Graham, 1995; Henderson-Sellers et al., 1996; Nikora and Munson, 2003; Ping et al., 2002; Zhao et al., 1998)
ClassStmt	Number of statements	Number of statements in the code (used as the size metric in our study)
MaxCyc	Maximum cyclomatic complexity	The highest McCabe complexity of a function within the class
ClassChg	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	Changesize	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 (CompFaults divided by the accumulated size of the modified classes in the component)

An additional benefit from using simple models is that they require less data to be collected, as 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. We selected univariate models based on code characteristics because, due to their simplicity and easiness of application, they can be considered as very basic prediction models that every other prediction method should be benchmarked against. It is so because any other prediction method will be at least equally expensive to apply, so in order to justify its application it would need to be more accurate than simple univariate regression.

In order to select the best single fault predicators from the class and the component metrics we perform a correlation analysis. The correlation analysis is commonly used for that purpose by other researchers (Ohlsson et al., 1997; Zhao et al., 1998). It quantifies the relation between two metrics as a value between -1 and 1. An absolute value of a correlation close to 1 characterizes good predicator variables. The values close to zero indicate a very weak linear relationship between the variables, and thus a low applicability of one variable to predict the other.

In this study we build two prediction models. One of them predicts faults at the class level and the other one predicts faults at the component level. The models are built using a univariate linear regression. The univariate linear regression estimates the value of the dependant variable (the number of faults or the fault-density) as the function of an independent variable (Nikora and Munson, 2003):

$$f(x) = a + bx \tag{1}$$

Even though our prediction models attempt to predict the actual value of fault density, in this study we use this information only as an indicator of the order in which the code units should be analyzed.

4.2. Expert estimation

The expected outcome of the expert estimation is a ranking of the code units according to their decreasing fault density. Such a ranking makes it possible to compare the accuracy of an expert estimation with the accuracy of a prediction made by our prediction models.

In the beginning of this study we have performed a number of interviews with our experts. The goal was to establish an appropriate level for performing 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 does, it was very difficult for them to predict the responsibility of particular classes within components. Additionally, the amount of data (249 classes for System A, and 180 for System B2) was considered unmanageable. The number of components is significantly smaller – there are 35 components with modified classes in System A and 43 in System B2. Therefore, in this study the expert estimation is performed only at the component level.

The expert estimation was performed individually by each of our experts. During the individual rankings the experts were provided with the list of modified components. Additionally, for each component, we enclosed the information concerning the subsystem to which the component belongs, as well as the accumulated size of the modified classes within the component. The experts were asked to rank the components according to their decreasing fault density. Additionally, for System A we managed to organize a consensus meeting. As input to this meeting we provided the experts with the individual rankings. The goal of the consensus meeting was to prepare a common "joint" ranking of components. In all cases, the experts were allowed to not rank all the components.

Before performing the estimations each of the experts individually was given a detailed description of the study in order to assure full understanding of the task. The experts were informed that their goal is to rank components according to their decreasing fault density, not according to the number of faults or their perceived severity. To further assure the understanding of the task the experts were also presented with a detailed description of how their estimations will be evaluated.

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 that they provide. We consider a prediction method better if, by following it, we are able to detect more faults by analyzing the same amount of code as compared to another prediction method. Therefore, we evaluate different predictions by plotting the percentage of faults that would be detected if analyzing a system according to a certain prediction method against the accumulated percentage of code that would have to be analyzed.

To obtain a point of reference for our evaluations, we introduce two reference models:

- Random model – the model describing a completely random search for faults.
- Best model – the theoretical model that makes only the right choices about which code unit to analyze first.

The Random model provides a baseline for evaluating our models, as it describes what results, on average, we could expect if we analyzed the code not following any model at all. The Random model is the same for all systems – on average by analyzing n% of code we find n% of faults. The Random model looks the same for the prediction at the class and at the component level.

The Best model provides a boundary of how good the prediction can be. In this theoretical model, the code units are selected according to their actual fault density. The Best model looks differently for different systems, because it depends on the actual distribution of faults in the system. The Best model is also different for predictions at the class and at the component level. The class level prediction has finer granularity and therefore, at least theoretically, it is able to provide more precise results. In this study we assess the practical value of having finer granularity prediction by comparing the Best model for components and for classes.

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

As described in Section 4.1, we begin building our prediction models with selecting the best individual fault predicator. We do that by performing a correlation analysis. In the correlation analysis we look for the best predicator of either fault density or the number of faults, as from the number of faults we can calculate the fault density by dividing the predicted number of faults by the size of a code unit (i.e., a class or a component). The correlation analysis is performed for both class and component level metrics. The class level metrics are explained in Table 1, and the component level metrics are explained in Table 2. The results of correlation analysis are presented in Table 3.

The highest correlations are marked in bold in Table 3. As it can be noticed, the most promising fault predicator for both classes and components is the size of the modification (ClassChg metric at the class level, and CompChg metric at the component level). 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., the class or component.

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

5.2. Expert estimation

5.2.1. Expert predictions concerning System A

In total, six experts performed predictions concerning System A. At first they performed ranking of the components individually. Later the group of experts was presented with the task of making one joint decision using individual results as input to the discussion. The distribution of "votes" of individual experts, the group consensus, and the actual ranks of the components are presented in Table 5. Only those components that were selected by at least one expert are presented.

In the individual rankings none of our experts ranked all 35 components. In fact, each expert ranked between 5 and 8 components. Altogether, the experts pointed out 15 different components, i.e., none of them had any opinion about the fault-proneness of the remaining 20 components. These 15 ranked components together account for about 60% of the code.

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

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 pinpointed them. These are components with numbers: 2, 4, 5, 6, 10, 12. The other group consists of components selected only by one or two interviewees, i.e., components with numbers: 1, 3, 7, 8, 9, 11, 13, 14, 15. It is quite clear that apart from two exceptions (components 2 and 6) there is a discrepancy between the ranks assigned by the 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.

Table 3
Correlation analysis results at the class and the component level

	Class level metrics								Component level metrics			
	Coup	NOC	WMC	RFC	Class	MaxCyc	DIT	LCOM	Class Chg	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

The highest correlations for respective levels (class, component) are marked in bold.

All components ranked in the "group consensus" ranking are the components that were selected by the majority of experts in the individual rankings. Components 2 and 6 were ranked according to the trend from the individual rankings, which was not surprising because the experts were quite consistent in ranking them as the first and the third in the individual rankings. Ranking components 10 and 4 as the second and the fourth, respectively, must have been an outcome of the group discussion, because such a ranking was not suggested by any individual expert.

5.2.2. Expert predictions concerning System B2

Five experts performed predictions concerning System B2. Their individual rankings together with the actual ranks of the components are presented in Table 6. Only those components that were selected by at least one expert are presented. Out of 43 components that were modified in System B2, the experts pointed out 16 components, i.e., none of our experts had any opinion regarding remaining 27 components. The 16 components selected by our experts account for about 66% of the code.

Similarly to the predictions concerning System A, in System B2 the ranked components can be divided into two subgroups. One subgroup consists of the components that were selected by the majority of experts, i.e., that were selected by at least 3 experts. To this group belong components with numbers: 2, 5, 6, 8, 12. The other subgroup consists of components that were selected by the minority of our experts. These are components with numbers: 1, 3, 4, 7, 9, 10, 11, 13, 14, 15, 16.

It seems that the agreement concerning the rankings was quite low among the experts that performed predictions in System B2. From the components selected by the majority of the experts, the highest agreement was achieved for the components with numbers 5 and 12. We see this agreement as weaker, as compared to the agreement concerning components 2 and 6 in System A. This is, however, our subjective judgment only.

5.3. Evaluation of prediction

Our prediction starts with building the reference models. For both systems 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).

The reference models created for System A are presented in Fig. 1. The reference models created for System B2 are presented in Fig. 2. As can be noticed, both graphs look similar, and some common conclusions can be drawn for both systems. In both systems, the Random model is quite far from the best possible model, which indicates that there

Table 4
Prediction models build in the study

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 * ClassChg + 0.018	FaultDensity = (0.002 * ClassChg + 0.018)/ClassStmt

"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.

Table 5
The rankings of individual experts and the joint ranking of all experts

	Component														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Expert1 ranking				1	4	2			3	5					
Expert2 ranking		3				1						2		4	5
Expert3 ranking	5	3		6	7	1				2		4			
Expert4 ranking		2		6	3	1	5			4		7			
Expert5 ranking		3	5		1	2		4	6			7			8
Expert6 ranking		4		1	2					5	7	8	3	6	

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. The components are presented in the order of their decreasing actual fault density.

Table 6
The rankings of individual experts concerning System B2

	Component															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Expert1 ranking	1		4		3							2				
Expert2 ranking		4			1	3	6		2	5						
Expert3 ranking					1	3		4				2				
Expert4 ranking		8	9	3	2	7		1			4	5	6	9	10	11
Expert5 ranking		2		4	5			1				3				

16 components out of 43 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. The components are presented in the order of their decreasing actual fault density.

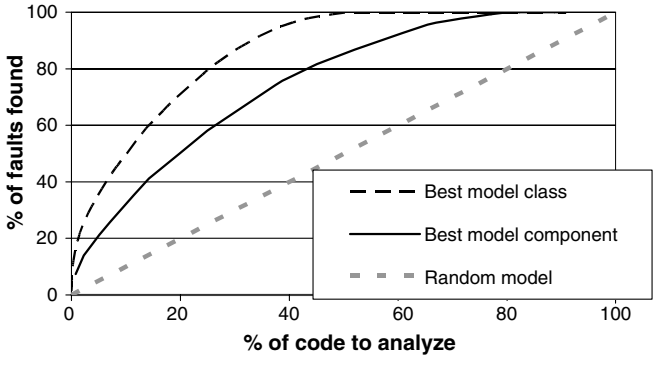

Fig. 1. Reference models in System A – an evaluation.

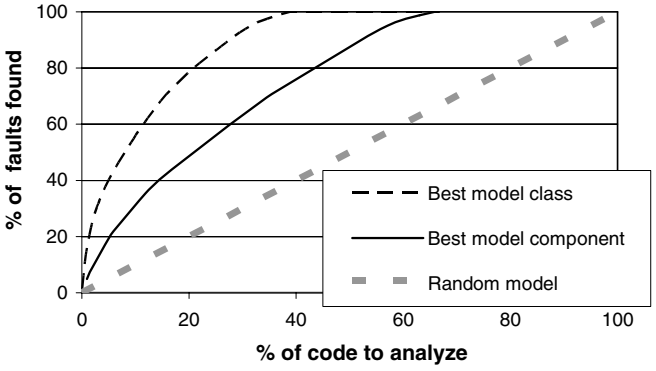

Fig. 2. Reference models in System B2 – an evaluation.

is large room for efficiency improvement that can be filled by an accurate fault prediction. For example, by analyzing 20% of the code randomly we can find 20% of faults. Ideally, in both systems by analyzing the most fault-prone 20% of the code we should be able to find up to 70% of the faults in case of the class level prediction (see Best model class in Figs. 1 and 2), and up to about 50% of the faults, if the prediction is made at the component level (see Best model component in Figs. 1 and 2).

Although theoretical, the higher maximum possible improvement achieved by predicting at the class level indicates that the class level prediction should be able to give better results. The class level prediction is made based on more fine-grained information and, therefore, it is more

precise. From Fig. 1 we see that, theoretically, the best component level prediction is capable of providing about two-third of the improvement over the random model offered by the best class level prediction (in Fig. 1 the distance between Best model component and Random model is more or less equal to 2/3 of the distance between the Best model class and the Random model). The gain from using a class level prediction is even more visible in System B2. In Fig. 2, we can see that the best component level prediction can be only half as good as the best class level prediction. The reader must bear in mind that this discussion concerns the best possible models that predict fault density at the respective code unit levels. It does not reflect the performance of our models.

The evaluation of expert estimations and our prediction models when applied to System A is presented in Fig. 3. In Fig. 3, we present all the individual expert estimations, "group consensus" estimation, both of our statistical prediction models (ClassPred, ComponentPred), and three reference models (Random model, Best model class, and Best model component).

From Fig. 3, we can conclude that both statistical prediction models clearly outperform the expert estimations. They not only offer higher accuracy in the range of code covered by any of the expert estimations (approximately up to 50% of code of System A) but also provide predictions that are significantly better compared to the Random model for the rest of the code. By comparing ClassPred with the best of the expert estimations for the percentage of code covered by the expert estimations we can see that ClassPred offers three times as large improvement over the Random model as the best of expert estimations.

Other findings from Fig. 3 concern the practical gain from using more fine grained information and predicting at the class level. As we can see there is a clear 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., compared to the Best model component.

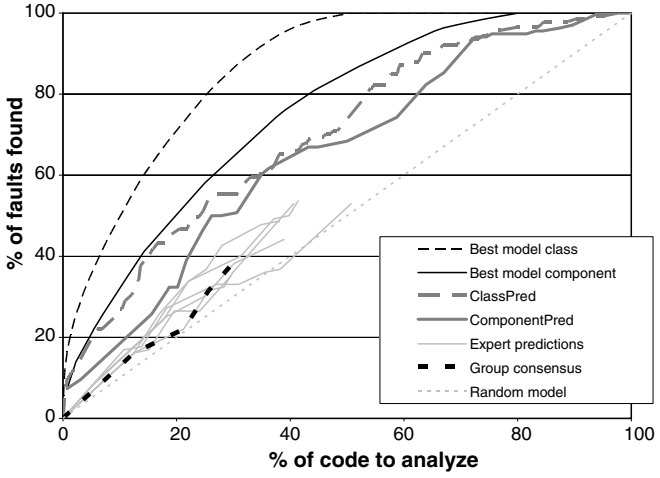

Fig. 3. Statistical prediction model vs. expert prediction in System A – the evaluation of accuracy.

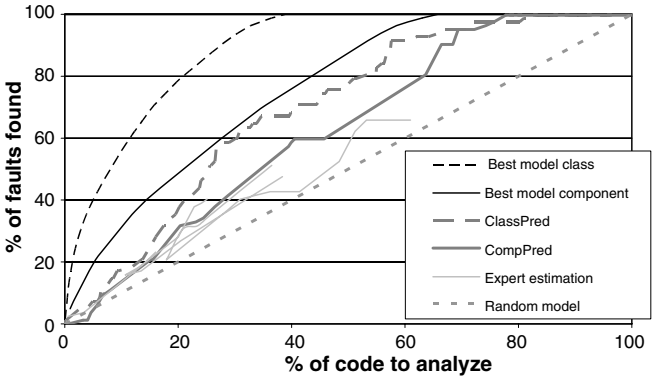

Fig. 4. Statistical prediction model vs. expert prediction in System B2 – the evaluation of accuracy.

Quite surprisingly the "Group consensus" estimation turns out to be one of the worst estimations made by our experts. Some of the individual estimations are actually not only more correct but they also account for more code.

The evaluation of expert estimations and our prediction models when applied to System B2 is presented in Fig. 4. As in the case of System A, we present all the individual expert estimations, both of our statistical prediction models (ClassPred, ComponentPred), and three reference models (Random model, Best model class, and Best model component).

In Fig. 4, we can see that for small percentages of the code (i.e., up to about 15% of the code) both statistical prediction methods and expert estimations provide equal gain over the Random model. When over 15% of code is analyzed, the gain from using the statistical prediction model is significantly larger compared to the gain from using any of the expert estimations. It is clearly visible in the case of the class level prediction (i.e., the ClassPred model). ClassPred provides a constant improvement over the Random model. It not only outperforms all expert estimations, but provides a significant improvement over the Random model for the range of code not covered by any of the expert estimations.

The CompPred model is visibly worse than ClassPred. It is, however, not worse than the best of expert estimations. Additionally, the CompPred model provides an improvement over the Random model even for the range of code not covered by the expert estimations.

6. Discussion

6.1. Findings

The results obtained in our study seem to support the idea of building fault prediction models. We have shown that statistical models have some advantages over human expert estimation. The biggest advantage of statistical models is that they are not negatively affected by the size

of the dataset. Therefore, statistical prediction models are able to estimate the fault-proneness of all code units even in large systems. Ranking all code units in a large system may be a difficult task for human experts. For example, our experts were quite confident when it comes to ranking the first couple of components. Beyond a certain number of components they admitted they would put remaining components in a random order.

The other advantage of prediction models is a direct effect of their ability to cope with large datasets. As we have shown, the statistical prediction models can successfully operate on fine-grained data, e.g., they can predict the fault-proneness of individual classes instead of predicting the fault-proneness of entire components. Our results indicate that human experts, irrespectively of their experience, may not be able to grasp large and complex system structures. This makes it difficult for human experts to make predictions at a low level of a system structure. At the same time we have shown that predicting at a low level brings not only theoretical but also practical benefits. In both systems, which we analyzed, the theoretical best prediction at the component level provides on average only about 40–60% of the improvement that can be offered by the best theoretical prediction at the class level (compare Best model component with Best model class in Figs. 3 and 4). The superiority of the class level prediction is also visible in practice. Our class level prediction model noticeably outperforms our component level prediction model in all cases.

There is also one more advantage of predication models, which we have 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 a form of e.g., a script that collects and processes all the required information automatically. An 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 another team of designers, the experts may simply not be available.

On the other hand, an expert estimation has some positive aspects, also not evaluated in this study. Our statistical prediction models predict faults, but do not classify them in any way. Naturally, not all faults are the same – some of them may be more difficult to find than the others. 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 these kinds of faults. Due to the lack of appropriate data we could not verify this hypothesis in our study.

Another benefit of the expert estimation can be its flexibility. The experts can take into account information that is not present in the statistical model. For example, in our statistical prediction models the size of the modification is considered to be the best fault predicator and all our models are based on it. However, it may happen so, that even though the change in the component is relatively small, there were a number of people involved in introducing it, which may make such a change more prone to faults compared to a change introduced by a single designer. Such rare, project-specific issues are likely to be captured by experts but it is very difficult to predict them in advance and incorporate them into a statistical prediction model. However, when analysing estimations of our experts, we have found a number of worrying factors. The experts do not agree with each other, they either select different components, or, if they select the same components, they estimate their fault density differently.

Another interesting observation that can be made when comparing the performance of experts in System A and in System B2 is that much longer experience with the product does not affect the accuracy of the expert prediction. Our experts involved in performing the estimations concerning System A have about five years longer experience with the product than the experts involved in the estimations concerning System B2. However, the accuracy of the expert predictions concerning both systems is not very different. In both systems the experts have selected a similar number of components. These components account for a similar percentage of code (see Sections 5.2.1 and 5.2.2 for the details concerning the expert prediction results). From Figs. 3 and 4 we can see that the fault detection efficiency improvement gained by using expert predictions is similar in both systems. Since it is unreasonable to assume that product related experience has no impact on the accuracy of fault prediction, the only possible conclusion is that there is some threshold value connected with experience, after which the accuracy of predictions is more or less similar. It is, however, important to remember that what we discuss here is a product related experience, not the experience as a whole. All our experts had experience in the development domain (i.e., telecommunications), which may additionally explain the similarity in their performance.

As a final remark we would like to make a methodological comment. As in can be observed in this section we believe that many of the problems our experts were facing were simply due to the size of the systems used in this study. Therefore, when conducting similar studies, we find it very important to assure that the size factor is present if the findings are to be applicable to large systems.

6.2. Validity

The reader must bear in mind that this paper has been meant more as an experience report than a formal experiment report. Our selection of projects was convenience-based – we have selected projects that were available to us. Also, since the entire exercise has not been performed as a controlled experiment, we can not assure that e.g., the experts did not have some at least partial knowledge about the actual fault-proneness of the components. For the reasons described in Section 3.2, and because of their poor performance, we believe this was not the case but we can not claim that we have eliminated this risk utterly.

The number of experts involved in each project may also be considered small and therefore it is difficult to perform any meaningful statistical analysis of their performance. However, there are a number of issues that make it easier to generalize findings from our study. The study was performed in an industrial setting. We used real, large telecommunication systems. Our experts had real experience and knowledge about the project. Additionally, they were motivated and interested in the study, which should have contributed positively to the quality of their predictions.

We also believe that our statistical prediction models obtained in this study are general. When building them, we followed the good academic practice of building models on different data than the data used to evaluate the models. We evaluated our models not only using the next release of the system the models were built on, but also using another system. We believe that all these factors make the evaluation of our statistical prediction models reliable. Additionally, the models built in our study share many similarities with models built by other researchers to predict faults and fault densities in modified code units. What we essentially do in our models is a prediction of the number of faults in the modified code using the size of modification, and prediction of fault-density in modified code units using the relative modification size. The same measures were suggested as the best predictors for respective purposes by other researchers (e.g., Nagappan and Ball, 2005; Selby, 1990).

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

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

7. Conclusions

The goal of this study was to compare the accuracy of fault predictions made by statistical fault prediction models with the accuracy of fault predictions made by human experts. We compared both prediction methods by applying them to two large software systems from the telecommunication domain. To perform the study we invited eleven experts involved in the development of these systems and we built two statistical fault prediction models. Our statistical fault prediction models were built based on data different from the data used in the evaluation.

The evaluation was performed from the perspective of an increase of the fault detection efficiency that could have been obtained if analyzing the code units in the order suggested by the experts or in the order suggested by our statistical models. Both prediction methods were evaluated against three reference models: a model based on a random selection of the code units for analysis, the theoretically best model for predicting faults at the class level, and the theoretically best model for predicting faults at the component level.

We found that both the expert estimations and the statistical prediction models provided an improvement over the random selection of code units for analysis. When comparing the performance of the expert estimations with the performance of the statistical models we found that the statistical prediction models outperformed the expert estimations. For example, for the systems that we analyze in this study, we find that for the portion of code covered by the expert estimations our statistical fault prediction models offered a higher improvement as compared to the best of the expert estimations. Moreover, the statistical predictions continued to provide an efficiency improvement over not using any model even after the point where our experts gave up.

We identified a number of reasons for the statistical models being better. Statistical models are not affected by the size of the dataset so they perform equally well on small and large systems, while the human ability to grasp the complexity of larger systems is limited. In addition, the ability to deal with large datasets makes it possible for the statistical models to perform more fine-grained predictions, i.e., predictions at a lower level. We showed that a more fine-grained prediction, e.g., a prediction at the class level instead of at the component level, is not only better from a theoretical but also from a practical perspective. Our class level prediction model was more accurate compared to our component level prediction model in both examined systems.

In this study we also discussed other advantages and disadvantages of statistical prediction models and expert estimations. The statistical prediction methods are reasonably cheap to build and apply, as well as they can be used in the absence of experts, e.g., when a project is transferred to another development organization. On the other hand, expert estimations are more flexible and can take into account some project specific issues that can affect fault-proneness of the components. Such project specific issues are usually hard to incorporate into otherwise general statistical fault prediction models.

Acknowledgements

The authors would like to thank Ericsson for providing us with the data, Ericsson staff members for active participation in this 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

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

Biffl, S., Grechenig, T., Kohle, M., 2000. Evaluation of inspectors' defect estimation accuracy for a requirements document after individual inspection. Proceedings of 7th Asia-Pacific Software Engineering Conference, 100–109.

Boehm, B.W., 1981. Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.

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

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

Cartwright, M., Shepperd, M., 2000. An empirical investigation of an object-oriented software system. IEEE Transactions on Software Engineering 26, 786–796.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20, 476–494.

Chidamber, S.R., Darcy, D.P., Kemerer, C.F., 1998. Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Transactions on Software Engineering 24, 629–639.

El Emam, K., Laitenberger, O., Harbich, T., 2000. The application of subjective estimates of effectiveness to controlling software inspections. Journal of Systems and Software 54, 119–136.

El Emam, K., Melo, W.L., Machado, J.C., 2001. The prediction of faulty classes using object-oriented design metrics. The Journal of Systems and Software 56, 63–75.

Fenton, N., Neil, M., 1999. A critique of software defect prediction models. IEEE Transactions on Software Engineering 25, 675–689.

Fenton, N., Ohlsson, N., 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software Engineering 26, 797–814.

Fenton, N., Pfleeger, S.L., 1997. Software metrics: a rigorous and practical approach. PWS, London, Boston.

Fioravanti, F., Nesi, P., 2001. A study on fault-proneness detection of object-oriented systems. Fifth European Conference on Software Maintenance and Reengineering, 121–130.

Graham, I., 1995. Migrating to Object Technology. Addison-Wesley Pub. Co., Wokingham, England, Reading, Mass.

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

Hughes, R.T., 1996. Expert judgement as an estimating method. Information and Software Technology 38, 67–76.

Khoshgoftaar, T.M., Seliya, N., 2003. Fault prediction modeling for software quality estimation: comparing commonly used techniques. Empirical Software Engineering 8, 255–283.

Khoshgoftaar, T.M., Allen, E.B., and Deng, J., 2000a. Controlling overfitting in software quality models: experiments with regression trees and classification. In: Proceedings of the 17th International Software Metrics Symposium. pp. 190–198.

Khoshgoftaar, T.M., Allen, E.B., Jones, W.D., Hudepohl, J.P., 2000b. Accuracy of software quality models over multiple releases. Annals of Software Engineering 9, 103–116.

Khoshgoftaar, T.M., Allen, E.B., Jianyu, D., 2002. Using regression trees to classify fault-prone software modules. IEEE Transactions on Reliability 51, 455–462.

Li, X., Liu, Z., Pan, B., Xing, D., 2001. A measurement tool for object oriented software and measurement experiments with it. 10th International Workshop New Approaches in Software Measurement. Springer-Verlag, Berlin, Germany, pp. 44–54.

Nagappan, N., Ball, T., 2005. Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th International Conference on Software Engineering ICSE 2005. pp. 284–292.

Nikora, A.P., Munson, J.C., 2003. Developing fault predictors for evolving software systems. Proceedings of the Ninth International Software Metrics Symposium, 338–349.

Ohlsson, N., Eriksson, A.C., Helander, M., 1997. Early risk-management by identification of fault-prone modules. Empirical Software Engineering 2, 166–173.

Ohlsson, N., Zhao, M., Helander, M., 1998. Application of multivariate analysis for software fault prediction. Software Quality Journal 7, 51–66.

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

Ostrand, T.J., Weyuker, E.J., Bell, R.M., 2005. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering 31, 340–355.

Pighin, M., Marzona, A., 2003. An empirical analysis of fault persistence through software releases. Proceedings of the International Symposium on Empirical Software Engineering, 206–212.

Pighin, M., Marzona, A., 2005. Reducing corrective maintenance effort considering module's history. Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, 232–235.

Ping, Y., Systa, T., Muller, H., 2002. Predicting fault-proneness using OO metrics. An industrial case study. Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, 99–107.

Selby, R.W., 1990. Empirically based analysis of failures in software systems. IEEE Transactions on Reliability 39, 444–454.

Shepperd, M., Cartwright, M., 2001. Predicting with sparse data. IEEE Transactions on Software Engineering 27, 987–998.

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

Thelin, T. 2004. Team-based fault content estimation in the software inspection process. In: Proceedings of 26th International Conference on Software Engineering. pp. 263–272.

Zhao, M., Wohlin, C., Ohlsson, N., Xie, M., 1998. A comparison between software design and code metrics for the prediction of software fault content. Information and Software Technology 40, 801–809.

Zhong, S., Khoshgoftaar, T.M., Seliya, N., 2004a. Analyzing software measurement data with clustering techniques. IEEE Intelligent Systems 19, 20–27.

Zhong, S., Khoshgoftaar, T.M., Seliya, N., 2004b. Unsupervised learning for expert-based software quality estimation. Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering, 149–155.