N. Ohlsson and C. Wohlin, "Experiences of Fault Data in a Large Software System", Special issue on Lessons Learned About Safety-Critical Systems in the journal of Failure and Lessons Learned in Information Technology Management: An International Journal, Vol. 2, No. 4, pp. 163-171, 1998.

Experiences of Fault Data in a Large Software System

Niclas Ohlsson and Claes Wohlin
Dept. of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
E–mail: (nicoh, clawo, maryh)@ida.liu.se

Abstract

Early identification of fault-prone modules is desirable both from developer and customer perspectives since it supports planning and scheduling activities that facilitate cost avoidance and improved time to market. Large scale software systems are rarely built from scratch, and usually involve modification and enhancement of existing systems. This suggest that development planning and software quality could greatly be enhanced, since knowledge about product complexity and quality of previous releases can be taken into account when making improvements in subsequent projects. In this paper we present results from empirical studies at Ericsson Telecom AB which examine the use of metrics to predict fault-prone modules in successive product releases. The results show that such prediction appears to be possible and has the potential to enhance project maintenance.

Tables wrongly numbered and text is missing. See paper copy!

1 Introduction

Failures in telecommunication systems can cause major disruptions and threaten critical services in society (Khoshgoftaar and Kalaichelvan, 1995), although telecommunications are not a normal safety-critical system. The quality of software has a major impact on the overall performance of telecommunication systems. It is well-known that a major part of software development cost is spent on maintenance (Henry and Kafura, 1981). It has also been reported that the cost for handling faults disclosed during testing and operation amount to large costs (Ohlsson et al., 19961996). In addition, it is well-established that the costs for fault correction grows with the number of phases between introduction and detection of faults (Boehm, 1981). Shen et al. (1985) concluded that schedule and/or resource constraints require that techniques which facilitate early detection are needed. A number of studies have aimed at predicting the most fault-prone modules at the completion of coding, see for example (Munson and Khoshgoftaar, 1992; Ebert and Liedtke, 1995). As these models can only be used to schedule testing activities when all modules have been implemented, models that are applicable earlier in the development phase are desirable. This is especially critical in projects where modules are completed at different time points, which appears to be common in most organisations, as existing prediction techniques require available data from all components. While the results from design are limited, there have been a few documented studies, see for example (Lennselius, 1990; Ohlsson et al., 1996). Still, even at the completion of design, the impact of the prediction is relatively limited. Such models can only be used to suggest extra inspection of fault-prone design modules, and to allocate more experienced people and time to those critical components. Therefore, models are needed that can be applied before the design, at the completion of the requirements specification process. These models can be based on previous knowledge or

information from the current system.
This paper presents a case study of fault data from two consecutive releases of a large telecommunication system (denoted n and n+1). In this context it is important to have clear interpretations of errors, faults, defects, failures and trouble reports. Thus, we would like to make the following distinction between them. Errors are made by humans, which may result in faults in the software. The faults may manifest themselves as defects during testing or failures during operation. Thus, faults can be interpreted as bugs in the software, defects are detected in testing and failures are the actual malfunction in an operational environment. In this paper we have used *fault-prone modules* to denote the modules with the highest number of bugs disclosed independent of whether the faults are disclosed during testing or operation. Furthermore, *defect–prone modules* denote the modules that account for the highest number of faults disclosed during testing, and *failure–prone modules* is used to denote the modules accounting for the highest number of faults disclosed during the first office application (site test, i.e. test under operational conditions) and in operation, see Figure 1. Faults are reported using trouble reports, and whether it is a defect report or failure report is determined by the actual time of discovery. It should also be noted that faults are only calculated once, i.e. duplicate trouble reports have been removed from the data set.

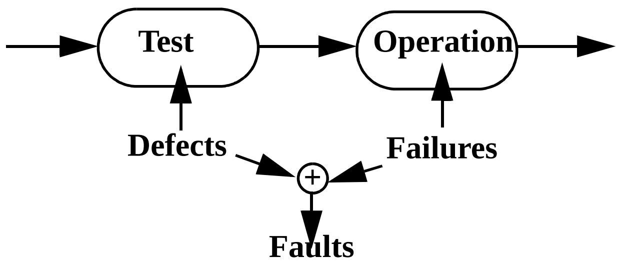

Figure 1. Illustration of defects, failures and faults.

The general objective of the study is to investigate methods of identifying fault–prone software modules, where failure-prone modules are of particular interest. The objective of this study was to investigate whether measurement from release n of the switching system could be used to identify the most fault-prone modules in release (n+1). Furthermore, the study investigated whether measurements from release n should be combined with those from release n+1 to see if the prediction models could be improved. Moreover, the goal is to use the knowledge acquired to improve the software development process in order to improve software quality in the future.

Some early results using parametric statistics have been reported in (Ohlsson and Alberg, 1996). The models have since been refined and analysed with non–parametric statistics (Ohlsson et. al., 1996). Identification of fault–prone modules has also been addressed by other researchers (Khoshgoftaar and Kalaichelvan, 1995) and (Munson and Khoshgoftaar, 1992). Few, if any, studies have exploited the opportunities to identify not only fault–prone modules in general, but actually failure–prone modules which are the main concern of the user. There is also a general lack of studies investigating whether identification of defect–prone modules means that we actually also identify failure–prone modules.

Another important issue is to establish when in the development phase we are able to identify modules which will be defect-prone and failure–prone. This paper investigates both identification of fault-prone modules between two releases and for failure-prone modules three different times for prediction are studied: history (previous release), the design phase and the test phase.

One important consideration is to address whether or not defect–prone modules are failure–prone. If defect–prone does not imply failure–prone, then we may have to improve the test methods.

The paper is organized as follows. In Section 2, the background to the study is presented. Section 3 presents an overview of the study, and Section 4 presents the analysis of the fault data collected. Section 5 discusses some of the results and experiences gained from the study. Finally, some conclusions are given in Section 6.

2 Background

2.1 Pareto principle

Quantitative methods for quality control and improvement have successfully been used within manufacturing for a number of years. Such approaches have also been applied within software engineering to enable a better understanding of software development and to improve software product quality. The Pareto principle, which guides improvements efforts towards the vital few and away from the trivial many, is one example of a quality improvement strategy. There exists a number of different examples of the Pareto principle applied within software engineering, see for example (Adams, 1984), (Munson and Khoshgoftaar, 1992), (Zuse, 1991), and (Schulmeyer and McManus, 1987). We have previously shown that the Pareto principle was supported by data from Ericsson Telecom AB (Ohlsson et al., 1996). Figure 2 illustrates that 20 percent of the modules in the two systems studied in this report were responsible for approximately 60 percent of all the faults.

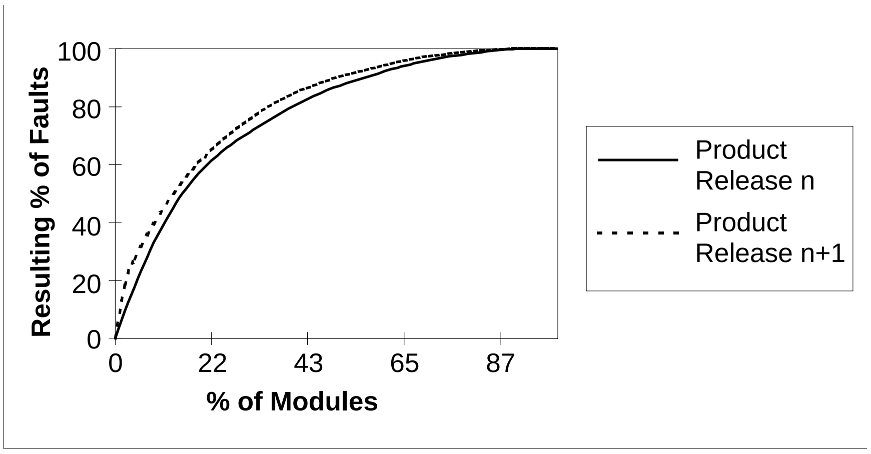

Figure 2. The Pareto diagram is based on data from two succeeding releases of a switching system developed by Ericsson Telecom AB and shows evidence of a 20-60 rule.

Developing models to predict the vital few fault-prone modules that are applicable early in the development process enables management to take special measures at an early stage. For example, more experienced engineers could be directed to support the development of critical parts, and additional and more extensive inspection and testing may be scheduled. Prediction of fault-prone modules at an early stage means reduced costs, since corrective maintenance in early phases are less expensive, and time to delivery may in fact be reduced as less re-work is required. Furthermore, identification of potential failure-prone modules is also important from a planning perspective, but even more important from a cost perspective.

2.2 Long-term study

This paper is part of a long–term empirical study conducted at Ericsson Telecom AB with the objective of studying how identification of fault–prone modules can be used to achieve cost–effective quality improvement. In release n of the system 130 modules have been analysed and in release n+1 232 modules have been investigated. Fault data have been collected from functional testing, system testing, first office application (i.e. the first 26 weeks and a number of site tests) and operation. It was possible to trace 69 modules developed for release n that were modified in release n+1. Release n+1 is a major system revision. Data are currently being collected for release n+2. The modules are of the size of 1000 to 6000 lines of code each.

The objective of our study was to distinguish between fault-prone and non-fault-prone modules, which suggests that discriminative analysis should be applicable. This was, however, particularly difficult for fault-prone, as discriminative analysis requires that there exists a threshold for classifying modules into fault-prone and non-fault-prone modules that is stable over releases. This turned out not to be the case. Instead we found that the threshold value (the actual number of faults) differed between the different releases although 20 percent of the modules were responsible for 60 percent of the faults. After careful examination of (Munson and Khoshgoftaar, 1992), we have found that this appears to be true for other data sets too. For failure-prone the threshold was no problem as it was decided to view any module with a failure as being failure-prone. The threshold was thus quite natural. Thus, in this paper we have used one failure as threshold for the dependent variable, i.e. modules with one or more failures are classified as failure–prone. The underlying analysis of design measures is based on ordinal analysis, as it allows for changing the threshold with regards to what are viewed as being fault–prone modules (Ohlsson et. al., 1996). Actual threshold–values are not recommendations; thresholds should be determined in individual projects on the basis of, for example, the level of criticality of the system and market requirements. The primary objective of the thresholds as presented in this paper is to illustrate the outcome when applying the methods for identification of failure–prone modules.

The goodness of different models for identification of fault-prone modules is investigated using an Alberg diagram, see below and (Ohlsson et. al., 1996). The predictability of the different models for identifying failure-prone modules is viewed in contingency tables and the kappa coefficients are calculated to measure the agreement in classification of the modules (Siegel and Castellan, 1988). The kappa coefficient is the ratio of the proportion of times that the classifications is correct to the maximum proportion of times that the classifications could be correct. If the classifications completely agree, then kappa=1; whereas if there is no agreement between the classifications, then kappa=0. Kappa will assume -1 if there is a perfect misclassification.

3 Fault study

3.1 Overview of the study

The study is divided into five parts:

1. Identification of fault-prone modules in release n+1 using measures from release n

   The objective of the first part is to build a prediction model using design measures and correlate them to the number of faults disclosed. The intention is to build the model for release n and then use the model for the following release. The model should be used to identify fault-prone modules in release n+1, hence the information can be used to plan effort and improvement activities. In particular, we would like to investigate if models built for one release can be used successfully to identify fault-prone modules in a consecutive release.

2. Identification of fault-prone modules in release n+1 combining measures from release n with early measures from release n+1

   In order to try to improve the models from the previous part, an approach where metrics from release n and n+1 are combined is studied. The objective is similar to that in part 1. The belief is that a combined model would improve the predictability of the models, since it takes information from the current release better into account.

3. Identification of failure–prone modules using data from a previous release

   This part is aimed at investigating whether the information from release n concerning defect and failure–prone modules is a good predictor of failure–prone modules in release n+1. More than 90 percent of the modules in release n had one or more faults. Therefore, it is infeasible to use one fault as a threshold. Thus, when defect–prone modules from release n is used to predict failure–prone modules in release n+1, a threshold of five defects is used for the independent variable as an indication of potential failure–prone modules. When failure–prone modules in release n are used as the independent variable, one failure is used as threshold.

4. Identification of failure–prone modules using design measures

   The initial objective was to build prediction models in release n for identification of failure–prone modules based on design measures, which then should be validated with data from release n+1. Due to variation in quality between the two releases this was not possible. Instead design metrics were only evaluated within release n+1. Only the best design measure is reported here, as the main objective is to investigate different opportunities to identify failure–prone modules rather than evaluate which measures are the best predictors. To the best of our knowledge there exists no empirical evidence that complexity values higher than a specific threshold would indicate either defect- or failure–prone modules. However, there are results suggesting a relative stable distribution in line with the Pareto principle (Ohlsson et al., 1996). Therefore, the threshold is based on the percentage of failure–prone modules in release n+1. That is, 29 percent of the modules in n+1 had one or more failures. Hence, this percentage value is used as a threshold for the design measures.

5. Identification of failure–prone modules from defect–prone modules

   The objective of this part is to investigate whether the defect–prone modules identified in release n and n+1 are good indicators of failure–prone modules in the two releases. This means that fault data from testing are used to predict failure–proneness during operation. The rationale for selecting thresholds is the same as in part 3.

To summarize, the main differences are the focus on faults respectively failures, and the point of time when prediction can be made. The three latter parts imply, for example, three different points of time in a project, namely: project start (part 3), design phase (part 4), and testing phase (part 5). It is important to remember that the sooner we are able to identify modules which are likely to be fault-prone, the sooner we can take appropriate measures to deal with them. For example, we can allocate the best people, intensify inspections or take other special improvement measures.

3.2 Data collected

The data collected from the two releases (n and n+1) were based on 69 modules, ranging in size from 1000 to 6000 LOC. The modules constituted a subset of a large system. The metrics from the different releases were distinguished by adding (n) and (n+1) to each metric name. The dependent variable, denoted Fault(n+1) (which is derived from the number of trouble reports), was collected from testing, 26 weeks during a number of site tests and operation. The actual number of faults considered depends on if defects or failures are studied. Note that the modules developed communicate with signals, see Turner (1993) for more details.

From release n the following metrics were collected:

- SigFF(n)—the number of new and modified signals (this metric is available after the first impact analysis conducted before design)
- Dec(n)—the number of decision nodes (this metric is available at the completion of design)
- Cond(n)—the number of condition nodes (this metric is available at the completion of design)
- FANin(n)—the number of receive-signals (this metric is available at the completion of design)
- FANout(n)—the number of send-signals (this metric is available at the completion of design)
- LOC(n)—the number of lines of code (this metric is available at the completion of implementation)
- Density(n)—the number of faults divided by the number of lines of code.

From release n+1 the following metrics were collected:

- R(n+1)—the number of receive-signals in SigFF.
- S(n+1)—the number of send-signals in SigFF.
- Dec(n+1)
- Cond(n+1)
- LOC(n+1)

The modification degree, denoted Mod(n+1), was also measured by dividing Cond(n) by Cond(n+1). SigFF is the sum of receive-, send- and transit-signals. Designers pointed out that transfer-signals never caused problems and were therefore believed to not affect the fault-proneness of a module. Thus, only receive- and send-signals were collected in release n+1.

4 Data analysis

4.1 Introduction

Our experience from industry indicates that the percentage of modules that management was willing to spend extra effort on differed from project to project. Therefore it was not possible to determine a threshold value to distinguish between fault-prone and non-fault-prone modules. Furthermore, the most relevant statistical methods were determined by permissible transformations of the data (Fenton, 1991). The type of metrics used in this study have been claimed to be of ordinal scale (Zuse, 1991), which suggests that non-parametric techniques should be used. A more detailed explanation of the application of non-parametric techniques for this study can be found in (Ohlsson et al., 1996). Spearman’s rank-order coefficient (Siegel and Castellan, 1988) was used to assess the variables to rank the predictability of fault-prone modules (p=0.001). The models for fault-proneness were further evaluated with Alberg diagrams (Ohlsson et al., 1996), which have the advantage of being graphical and of making different types of errors visible for all thresholds at once. To evaluate the goodness of the predictions, the prediction errors must be considered. This includes two different types of errors: failing to identify failure–prone modules and identification of modules as failure–prone when they are not. These are hereafter referred to as errors of Type I and II respectively. It should be noted that a correct identification means actually pin–pointing a certain module correctly.

The diagrams also indicate whether the modules indicated as fault-prone, but are non-fault-prone, are relatively close to the threshold. In other words, if we have a threshold of five faults, Type II errors identifying modules with zero faults will generate a bigger gap than Type II errors identifying modules with four faults. The latter Type II error is of course less critical, especially if the corresponding Type I error excluded a module with five faults. For further discussion see (Ohlsson et al. 1996; Ohlsson, 1996).

For failure-prone modules, a threshold approach was adopted since it was quite natural to define failure-prone modules as modules with a failure. Thus, when identifying failure-prone modules, another approach taken than when identifying fault-prone modules. This is an important experience from the study conducted.

4.2 Fault-prone modules from in release n+1 using measures from release n

Spearman’s correlation coefficient was calculated for the correlation between a number of design metrics from release n and faults from release n+1, i.e. Faults(n+1) is used as the dependent variable. The result is displayed in Table 1. The analysis indicates that the most fault-prone modules in release n will also be most fault-prone in the successive release n+1.

Table 4–1.

	Fault(n+1)		Fault(n+1)		Fault(n+1)
Fault(n)	0.6541	Density(n)	0.6053	Dec(n)	0.4792
FANin(n)	0.6328	FDL(n)	0.5767	Cond(n)	0.4738
SigFF(n)	0.6296	FANout(n)	0.5754	LOC(n)	0.3760

To determine whether it would be profitable to combine certain variables into more complex models the correlation was calculated between pairs of the independent variables. FANin(n) showed low correlation to Fault(n) and Density(n), and it was therefore assumed to be profitable to combined these. The correlation between the dependent variable and FANin(n)*Fault(n) as well as the correlation between the dependent variable and FANin(n)*Density(n) was 0.74.

4.3 Identification of fault-prone modules in release n+1 combining measures from release n with early measures from release n+1

The next step was to determine whether it would be profitable to combine the metrics from release n and n+1. Spearman’s correlation coefficient was calculated for the metrics from release n+1, using Fault(n+1) as dependent variable. The correlation values are listed in Table 2, which also includes the Mod(n+1) variable. It should be noted that the correlations are about the same when using design metrics for release n+1 as when using design metrics from the previous release. That is, it appears that the design base, or the history of the modules, is important when we try to explain the fault-proneness.

Table 4–2.

	Fault(n+1)		Fault(n+1)		Fault(n+1)
S(n+1)	0.5448	Dec(n+1)	0.5850	Mod(n+1)	-0.3410
R(n+1)	0.4608	Cond(n+1)	0.5823		

To determine which variables could be combined with the variables from release n the correlations between the metrics were calculated. Cond(n+1) was selected to be combined with FANin(n) and S(n+1) with both FANin(n) and Fault(n). The former had a correlation of 0.7380 and the latter two 0.6511 and 0.7300. This shows that the combined approach may improve the correlation and hence the predictive ability of the models.

4.4 Evaluation of the best models for predicting fault-prone modules

The evaluation of the prediction models using an Alberg diagram suggested that a threshold of 20 percent could be used to identify modules responsible for 46 percent of the faults based on release n data. The best predictor available early in release n+1 was Dec(n+1), which identified 55 percent of the faults with a threshold of 20 percent. The Alberg diagram indicated that the best predictor model, using a 20%-threshold, would result in the identification of 59 percent. This model was, however, not identified by calculating the correlation values and on this basis combining the variables into more complete models. Instead the model was identified when analysing the result from the Alberg diagram, see Figure 3. Although this highlights the need for better methods for building more complex models based on ordinal data, the actual models show the applicability of prediction models to identify fault-prone modules: before the project has started, in the early analysis phases, and at the completion of design.

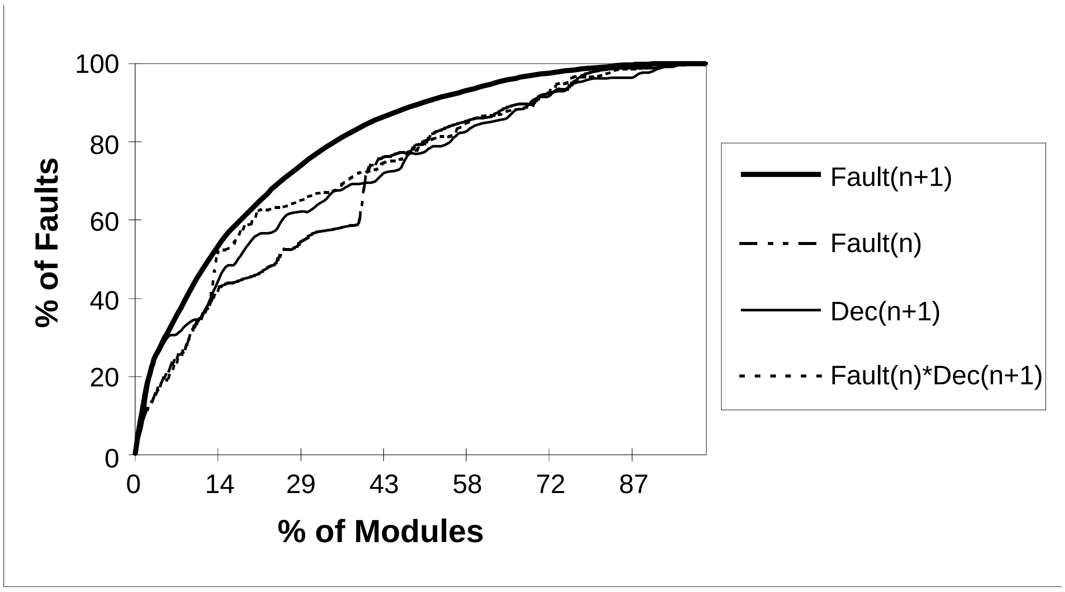

Figure 3. The Alberg diagram showing some of the best prediction models. The prediction models have a distance of 4% at 20%-threshold, and 10% at 30%-threshold.

4.5 Failure–prone modules from history

For software systems, it is normal practice that a system is regularly upgraded and released in new versions. This implies that some parts of the system are the same in different releases.

This information can be used to apply experience from one release to the next release or following releases. In this empirical study, the hypothesis is that defect- or failure-prone modules in release n are likely candidates for being failure–prone in release n+1. It was possible to trace 69 modules developed for release n that were modified in release n+1. The data from the historical analysis are shown in Table 3. It should be noted that only four modules were failure–prone in release n, see analysis A, while 18 modules were failure–prone in release n+1.

Table 4–3.

	Analysis A[a]		**Analysis B**[b]		**Analysis C**[c]	
	Threshold=1		**Threshold=5**		**Threshold=5**	
	Failure(n)		**Defect(n)**		**Fault(n)**	
Actual	F	Not F	F	Not F	F	Not F
Failure-prone(n+1) (18 observation)	4	14	14	4	15	3
Not Failure-prone(n+1) (51 observations)	0	51	28	23	28	23
Total observations	4	65	42	27	43	26
Misclassifications of Type I and II	78% (14/18)	0% (0/51)	28% (4/18)	55% (28/51)	17% (3/18)	55% (28/51)
Overall misclassifications	20% (14/69)		46% (32/69)		45% (31/69)	

a. Kappa 0.30
b. Kappa 0.16
c. Kappa 0.32

Analysis A in Table 3 illustrates that even though the Type I error is as high as 78%, there is no Type II error. This means that the modules that are failure–prone in release n are all failure–prone in release n+1. Possible explanations for this are the actual type of failure and late erroneous defect correction in test.

For analyses B and C, we have used five defects (Analysis B) or faults (Analysis C) as a threshold for the independent variable. It has earlier been suggested (Khoshgoftaar and Kalaichelvan, 1995) that this should be used as threshold for fault–prone modules. The threshold could therefore indicate failure–proneness. Using one defect or fault is not reasonable since this would identify 63 modules as being failure–prone. Even with a threshold of five defects in analysis B as many as 61 percent (42/69) of the modules are identified in release n as failure–prone. However, only 78 percent (14/18) of all the failure–prone modules in release n+1 are identified. Therefore, defect–prone modules in release n are poor predictors of failure–prone modules in n+1. This is also true for analysis C.

Another possible alternative would be to select a threshold based on the percentage of failure–prone modules in release n+1, i.e. assuming that this proportion of defect- and failure–prone modules will be stable over later releases. The number of potential failure–prone modules would be more realistic using 26 percent (18/69) as a threshold. However, only 28 percent of the failure–prone modules would be identified. This also holds for analysis C. Therefore, the two models in analyses B and C are not applicable.

4.6 Failure–prone modules from design measures

Earlier studies (Ohlsson et al., 1996) have indicated that models built on design metrics are worthwhile when the number of faults are considered as the dependent variable. Thus, it is reasonable to try this approach for failure–prone modules. In this study, fourteen different design measures are used to build prediction models for release n+1. Spearman's correlation coefficient (Siegel and Castellan, 1988) was used for a first analysis. All potential variables have low correlation values (below 0.35). There was, however, a rather low correlation among some of the variables, hence it could be possible to improve the model by combining the variables into more complex models. Multiplicative aspects of the potential variables will be investigated in later studies. In this particular case, the best design measure predictor was FANin, which is the number of input–signals for a module in the design. The result was later compared with lines of code, which was found to be doing even worse.

It has been suggested that prediction models should first be developed for one release, validated in the succeeding release, and then applied in the third release. However, the quality of the two releases varied widely, and it was therefore not possible to do so in this study. From a modelling point of view, the number of failure–prone modules in release n was too few. Instead, the explanatory ability of design metrics was evaluated by building the best possible model based on data in release n+1. The results shown in Table 4 are based on a threshold of one failure, which corresponds to 29 percent of the modules.

Table 4–4.

	Analysis[a]	
	FANin(n+1)	
Actual	F	Not F
Failure-prone(n+1) (67 observation)	28	39
Not Failure-prone(n+1) (165 observations)	39	126
Total observations	67	165
Misclassifications	58% (39/67)	24% (39/165)
Overall misclassifications	34% (78/232)	

a. Kappa 0.18

From Table 4, it can be seen that the explanatory ability is unsatisfactory, i.e. the misclassification is too high, including a large proportion of both Type I and II errors. This, in combination with the fact that the quality of the two releases differed, suggests that more complete models should be investigated, for example including verification effort and quality.

4.7 Failure–prone modules from defect–prone modules

The data from the testing phase can be used for both releases to predict the failure–prone modules. The problem with choosing relevant thresholds, discussed in respect to part 1, is relevant for this part, too. The results of the analyses are shown in Table 5, using a threshold of five defects for the independent variable.

Table 4–5.

Analysis n[a]		
	Defect(n)	
Actual	F	Not F
Failure-prone(n) (13 observation)	5	8
Not Failure-prone(n) (117 observations)	77	40
Total observations	82	48
Misclassifications	62% (8/13)	66% (77/117)
Overall misclassifications	65% (85/130)	

Analysis n+1[b]		
	Defect(n+1)	
Actual	F	Not F
Failure-prone(n+1) (67 observation)	47	20
Not Failure-prone(n+1) (165 observations)	102	63
Total observations	147	83
Misclassifications	30% (20/67)	62% (102/165)
Overall misclassifications	53% (122/232)	

a. Kappa -0.08

b. Kappa 0.06

The misclassification is also too high in this analysis. This means that modules that are defect–prone during testing are not failure–prone. A possible explanation is that other types of defects are discovered in operation, such as performance problems, that are difficult to test. This explanation is supported by experienced developers from Ericsson. This could also explain the result in part 3. A possible explanation of the fact that failure–prone modules in n are failure–prone in n+1 could be that modules which are critical from a capacity perspective in release n, will remain so in release n+1. The results indicate the need for a better understanding of the types of faults that result in failures and the types of the failures themselves. The results also stress the need to identify factors causing the defects which result in failures. Increased understanding is essential for quality improvement.

From all three studies regarding failure-prone modules, it can be noted that the kappa value is rather low. This indicates that it is difficult to find models that identify failure-prone modules based on solely product measures. Thus, other ways of identifying failure-prone modules must be sought.

5 Discussion and future work

A majority of the studies aimed at predicting the most fault-prone modules have focused on building prediction models applicable at the completion of the implementation phase. Such models may be difficult to use to improve allocation of test efforts as it is common that modules in real projects are not completed at the same time, which is a prerequisite for building such models. Therefore we claim that it is important that research resources are spent on trying to make predictions earlier, before coding has started, or even earlier. This should be worthwhile as such models would not only improve fault detection, e.g. by cost effective inspection and testing, but also fault avoidance, by improving allocation of implementation resources in terms of effort and skills. In this paper we looked at how such models can be improved by including historical data, e.g. fault-proneness in earlier releases and degree of modification. The results described in this paper indicate that it can be profitable to use data from earlier releases to predict the most fault-prone modules. The result also suggests that historical data give an early indication of which modules will be the most critical ones, in particular from a cost perspective. Based on our experience from building such models we have identified a number of areas that need more examination. First, there exists no praxis for how variables can be combined into more complete models without violating the transformations admissible for ordinal data. Existing techniques, such as principal component analysis (Khoshgoftaar and Kalaichelvan, 1995), require standardised data, which restrict the transferability of models to other data sets.

Future work should therefore focus on developing methods for analysing additive and multiplicative effects of several variables of ordinal type, as well as how to weight variables differently. Second, it is difficult to get manageable data sets, without a high risk of excluding the right data set, using present screening techniques, applicable to ordinal data. Third, the objective of early identification of fault-prone modules is to enable improved fault detection in terms of inspection and testing, and fault avoidance in terms of improved allocation of time and skills. Unless these parameters are included in the prediction model, the models will wear out. Therefore future work should aim at identify other attributes that should be included in the models. Fenton et al. (1995) developed a four-layered approach that could be used to develop such a more realistic and complete model that includes attributes of not only the product, but also process and resources. Fourth, a number of studies are only based on defect data from testing, omitting the actual performance of the system in usage. Such models will only be good for predicting the modules for which the test strategy used is likely to find many defects, not the modules which will be fault-prone or failure-prone, i.e. likely to have many faults associated that are disclosed during operation. Fifth, little work has been done on evaluating models predictability by building models for one release or project and then trying to evaluate predictability by testing on data from another project. Such validation is crucial to understand model applicability in a real industrial environment. Finally, surprisingly few studies have reported failure in achieving their objectives. It is our belief that there exists a significant number of studies which have not achieved their first stated objectives, and which failures, if published, would be of benefit to the research field.

6 Conclusions

In this paper we have investigated the opportunity to predict fault–prone modules based on design, code and fault data from two succeeding releases. The study revealed that failure–prone modules in release n are failure–prone in n+1. Other suggested independent variables are poor predictors of failure–proneness. However, this is not the same as saying that they do not explain any of the variation. It only means that on their own they are poor explanatory factors. Instead, the study suggests that methods that combine these different independent variables are needed. The results are, however, promising regarding prediction of fault-prone modules, although there is still room for improvement.

In this study, we have addressed two consecutive releases of a software system. This is an important aspect as in most cases it is not possible to both build, validate and use a prediction model within one release. It is, thus, important to investigate how to build models in one release, validate the model in the next release and then use the model in the third release. The transferability of a model between a software system's releases is crucial to success in the mission of identifying fault–prone modules prior to the operational phase.

A major problem with predictions is that failures are dynamic, hence it may be difficult to identify failure–prone modules using static measures. This is an issue which has to be further studied. One potential solution would be to take the use of modules into account when predicting failure–proneness. This would allow for capturing the dynamic aspects of usage in the independent variable.

Another important issue which has been addressed here is the point of time when we are able to identify fault–prone modules. To improve the usefulness of the predictions, they should preferably be done at an early stage. In this study, we have focused on data from the previous release, the design and the test phase. The knowledge from the previous release is important in identifying fault–prone modules, but this is not a feasible approach for new modules. Thus, it is very important to find early indicators of fault–proneness, since this is the only way to enable us to address the problem within the same release.

Models which identify fault–prone modules are important not only in enabling prediction during the operational phase, but also as a planning and control tool during development. Managers may use these models to improve the resource allocation for design, both in terms of effort and experience. Furthermore, knowing which modules are most likely to be failure–prone in operation suggest that the modules will be tested and inspected differently. Therefore more attributes need to be considered and incorporated in the models, for example verification effort and quality, in line with Fenton et al. (Fenton et al., 1995), to explain the variation and to be able to apply the models in subsequent releases.

Future work should not only aim at building these more complete models, but also aim at investigating additive and multiplicative aspects of design measures and measures from different phases, in order to gain more knowledge about how such a component fits into a more complete model. The results in this study also suggest that prediction models that are only based on test data will have limited applicability in real projects aiming at addressing operational issues.

Acknowledgements

The authors would like to thank Ericsson Telecom AB for supporting this empirical study.

References

Adams, E. (1984). “Optimizing preventive service of software products“. IBM Research Journal, 28(1):2-14.

Boehm, B. W. (1981). Software Engineering Economics. Prentice-Hall.

Ebert, C. and Liedtke, T. (1995). “An integrated approach to criticality prediction. In Proceedings of The Sixth International Symposium on Software Reliability, pp. 14-23, Toulouse, France.

Fenton, N., Neil, M., and Ostrolenk, G. (1995). Metrics and models for predicting software defects. Technical Report CSR/10/02, Centre for Software Reliability, City University, UK.

Fenton, N.E. (1991). Software Metrics-a Rigorous Approach. Chapman & Hall, London.

Henry, S. and Kafura, D. (1981). “Software structure metrics based on information flow“. IEEE Transactions on Software Engineering, 7(5):510-518.

Khoshgoftaar, T. M. and Kalaichelvan, K. S. (1995). “Detection of fault-prone programs modules in a very large telecommunication system“. In Proceedings of The Sixth International Symposium on Software Reliability, pp. 24-33, Toulouse, France.

Lennselius, B. (1990). Estimation of software fault content for telecommunication systems.

Technical report 104, Department of Communication Systems, Lund Institute of Technology, Lund University, Sweden.

Munson, J. C. and Khoshgoftaar, T. M. (1992). “The detection of fault-prone programs“. IEEE Transactions on Software Engineering, 18(5):423-433.

Ohlsson, N. (1996). Software Quality Engineering by Early Identification of Fault-prone Modules, Licentiate Thesis No 575, Dept. of Computer and Information Science, Linköping University, Sweden.

Ohlsson, N. and Alberg, H. (1996). Predicting fault-prone software modules in telephone switches. IEEE Transactions on Software Engineering, 22(12).

Ohlsson, N., Helander, M., and Wohlin, C. (1996). “Quality improvement by identification of fault-prone modules using software design metrics“. In Proceedings of the Sixth International Conference of Software Quality, pp. 1-13, Ottawa, Canada.

Schulmeyer, G. G. and McManus, J. I., editors (1987) Handbook of Software Quality Assurance. van Nostrand Reinhold Company.

Shen, V. Y., Yu, T.-L., Theabaut, S. M., and Paulsen, L. R. (1985). “Identifying error-prone software - an empirical study“. IEEE Transactions on Software Engineering, SE-11(4):317-323.

Siegel, S. and Jr., N. J. C. (1988). Nonparametrics Statistics for the Behavioural Sciences, McGraw-Hill, second edition.

Turner, K.J., editor (1993). Using Formal Description Techniques - An Introduction to ESTELLE, LOTOS and SDL. John Wiley & Sons.

Zuse, H. (1991). Software Complexity-Measures and Methods. Walter de Gruyter.