References

Basili, V., and Rombach, D. 1988. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14(6): 758–773.

Basili V. and Weiss, D. 1984. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering* 10(11): 758–773.

http://www.iese.fhg.de/Services/Projects/Public-Projects/Cemp.html

Hoisl, B., Oivo, M., Rombach, D. H., Ruhe, G., van Latum, F., and van Solingen, R. Shifting to goal-oriented measurement in industrial environments. Submitted for publication.

Morasca, S., Macchi, F., Grigoletti, M., and Gusmeroli, C. Goal-driven measurement in a maintenance project. Tech. Report n 96-047, Politecnico di Milano Dipartimento di Elettronica. Submitted for publication.

Early Risk-Management by Identification of Fault-prone Modules

NICLAS OHLSSON nicoh@ida.liu.se
Dept. of Computer and Information Sc., Linkōping University, S-581 83 Linkōping, Sweden

ANN CHRISTIN ERIKSSON ann-christin.eriksson@uab.ericsson.se
Ericsson Utvecklings AB, AXE Systems Management, S-125 25 Älvsjö, Sweden

MARY HELANDER maryh@ida.liu.se
Dept. of Computer and Information Sc., Linkōping University, S-581 83 Linkōping, Sweden

1.0 Introduction

Failure in telecommunication systems can cause major disruptions and threaten critical services in society (Khoshgoftaar and Kalaichelvan, 1995). 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., 1996). 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., 1996b). 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. It is also of interest to investigate to what extent present models, such as those described by Ohlsson et al. (1996b), can be improved by incorporating facts about the design-base quality and complexity.

This paper describes an empirical study conducted at Ericsson Telecom AB. Quantitative data presented was collected from two successive, large-scale switching software projects. 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.

2.0 Background

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., 1996b). Figure 1 illustrates that 20 percent of the modules in the two systems studied in this report were responsible for approximately 60 percent of all the trouble reports.

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.

3.0 Data Collected

The data collected from the two releases (n and $n + 1$) was 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 $TR(n + 1)$, was collected from testing and

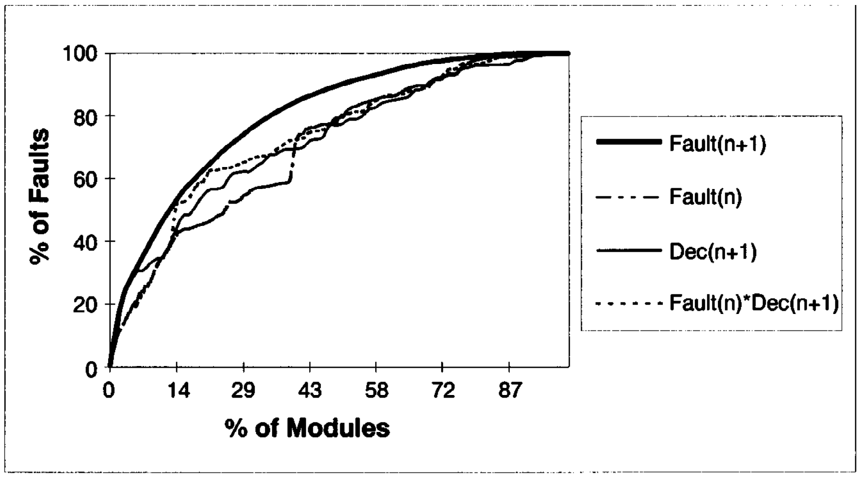

Figure 1. The Pareto diagram is based on data from two succeeding releases of a switching system developed by Ericsson Telecom AB and shows evidence of a 20-60 rule.

from 26 weeks during a number of site tests. 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)
- TR(n)—the number of trouble reports (this metric is available after 26 weeks at site test)
- Density(n)—the number of trouble reports 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.

4.0 Data Analysis

The objective of our study was to distinguish between fault-prone and non-fault-prone modules, which suggests that discriminative analysis should be applicable. However, 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 differed between the different releases even though 20 percent of the modules were responsible for 60 percent of the faults. This was also supported by data published elsewhere (Munson and Khoshgoftaar, 1992). Our experience from industry also indicated 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., 1996b). *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 were further evaluated with Alberg diagrams (Ohlsson et al., 1996), which have the advantage of being graphical and of making Type I and Type II errors visible for all thresholds at once. 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, 1996a; Ohlsson et al., 1996b).

Table 1. Spearman's ranking coefficient for metrics from release n.

	TR(n + 1)		TR(n + 1)		TR(n + 1)
TR(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

Table 2. Spearman's ranking coefficient for metrics from release n + 1.

	TR(n + 1)		TR(n + 1)		TR(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		

4.1 Analysis 1—Prediction Based on Release n

Spearman's correlation coefficient was calculated for the metrics from release n, using TR(n + 1) 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 successsive release n + 1.

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 TR(n) and Density(n), and it was therefore assumed to be profitable to combined these. The correlation between the dependent variable and FANin(n)*TR(n) as well as the correlation between the dependent variable and FANin(n)*Density(n) was 0.74.

4.2 Analysis 2-Prediction Based on Early Metrics 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 TR(n + 1) as dependent variable. The correlation values are listed in Table 2, which also includes the Mod(n + 1) variable.

It is notable that correlations are overall slightly higher for the metrics displayed in Table 1. That is, it appears that the design base, or the history of the modules, is important when we try to explain the fault-proneness. 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 TR(n). The former had a correlation of 0.7380 and the latter two 0.6511 and 0.7300.

4.3 Evaluation of the Best Models using Alberg Diagram

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

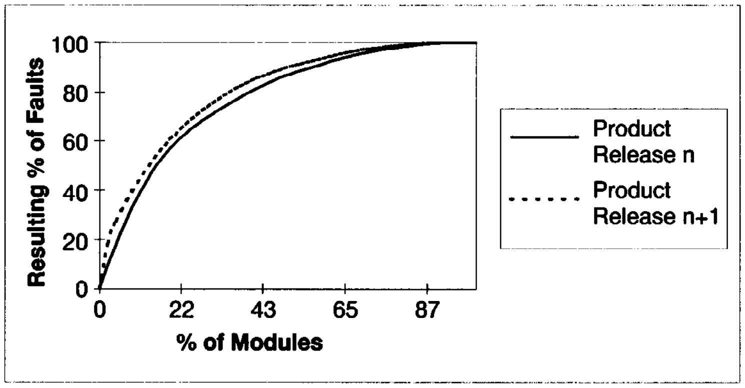

Figure 2. The Alberg diagram showing some of the best prediction models. The prediction models have a distance of 4% at 20%-threshold, and 10% at 30%-threshold.

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. Eventhough 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.

5.0 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 further. 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 release and modification degree. 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 gives an early indication of which modules will be the most critical ones. 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 et al., 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 fault 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 faults, not the modules which will be fault-prone or failure-prone, i.e. likely to have many faults associated that are disclosed during operation. Fifty, little work has been done on evaluating models predictability by building models for one release or project and ten 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.

References

Adams, E. 1984. Optimizing preventative service of software products. *IBM Research Journal* 28(1): 2–14.

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

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*. London: Chapman & Hall.

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 Kalaichelyan, K. S. 1995. Detection of fault-prone programs modules in a very large telecommunication system. *The Sixth International Symposium on Software Reliability* Touluse, France, 24–33.

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. 1996a. 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., Helander, M., and Wohlin, C. 1996b. “Quality improvement by identification of fault-prone modules using software design metrics. *Sixth International Conference of Software Quality* Ottawa, Canada, 1–13.

Schulmeyer, G. G., and McManus, J. I., eds. 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., ed. 1993. *Using Formal Description Techniques—An Introduction to ESTELLE, LOTOS and SDL*. John Wiley & Sons.

Problems and Prospects in Quantifying Software Maintainability

JARRETT ROSENBERG jarrett.rosenberg@sun.com; jarrett.rosenberg@acm.org
Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043

In one form or another, quantifying the maintainability of software has been attempted for decades, in pursuit of two goals:

- predicting the likelihood of corrective maintenance,
- predicting the difficulty of maintenance activities, corrective or otherwise.

Despite the extensive activity in this area, we are not much further along than when we started. This paper looks at some of the reasons why, and the factors necessary for future progress.

Goal #1: Predicting Likelihood of Corrective Maintenance

Predicting the likelihood of defects in source code has long been a major focus of software metrics research (Melton, 1996). There are two necessary factors that must be considered in any such prediction: characteristics of the source code itself (including the design it embodies), and characteristics of the testing process used on the code up to the point when the prediction is made. Let us examine each of these in turn.

Source Code Characteristics

Some thirty years of research has yielded two basic classes of metrics: size metrics (most famously, lines of code and the Halstead metrics) and complexity metrics (most famously, McCabe's cyclomatic complexity metric). Size metrics do predict likelihood of defects, but in precisely the same way as any exposure variable does: the more source code, the more defects. If an application's source code were to be evenly divided into equal-sized files, size would no longer accurately predict likelihood of defects per file. For this reason, size metrics are only useful as covariates in evaluating the effect of other, non-size, metrics, that is to say, predictions must be adjusted for size to be valid.