Quality Improvement by Identification of Fault-Prone Modules using Software Design Metrics

Ohlsson, N., Helander, M. and Wohlin, C.

Accepted for publication at
the Sixth International Conference on Software Quality,
Ottawa, Canada, October 1996.

Accepted at: *Sixth International Conference on Software Quality*, Ottawa, 28-30 October, 1996.

Quality improvement by identification of fault-prone modules using software design metrics

Niclas Ohlsson, Mary Helander & Claes Wohlin

Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden
Phone: +46-(0)-13-28 25 94
E-mail: nicoh@ida.liu.se, maryh@ida.liu.se & clawo@ida.liu.se

Abstract- Quality improvement in terms of lower costs, shorter development times and increased reliability is desired from both the developer's and their customer's point of view. To enable early identification of problems, and subsequently to support planning and scheduling, methods for identification of fault-prone modules are desirable. This paper presents a case study of how design metrics were used at Ericsson Telecom AB to identify fault-prone modules at the design phase. Derivation of a model for identification of fault-prone modules is discussed, with emphasis on the use of appropriate statistical methods. The model derivation and statistical methods are exemplified using data from two releases of a large software system. The model is derived from the first release and its ability to pinpoint the most fault-prone modules is evaluated for the second release. It is concluded from the empirical study that it is possible to identify a small portion of modules contributing to a large number of faults. For example, in the (n+1)th release, 20% of the modules were responsible for 57% of the total number of faults. The strategy used in this paper resulted in a prediction model that would have identified 49% of the total number of faults. This suggests that collection of design metrics and using them for identification of fault-prone modules to support quality improvement is certainly worthwhile.

1 INTRODUCTION

Reduced development costs, shortened development time, and increased reliability are not only desired by most software developers, but are also demands of the customer. It is well-known that the costs for fault correction grows with the number of phases between the introduction and detection of faults (Boehm, 1981). In particular, failures in the operational phase are very expensive. Thus, methods for early detection of fault-prone modules are highly desirable, since they allow for better resource planning and scheduling, in addition to cost avoidance by effective verification.

This paper focuses on identification of fault-prone modules based on design metrics. The objective is to identify design metrics which are good indicators of fault-proneness. Similar approaches have been reported for code metrics in, for example Munson and Khoshgoftaar (1992), and Ebert and Liedtke (1995). While the results from design are limited, there have been a few documented studies, see for example Lennselius (1990); and Khoshgoftaar et al. (1996). Since these results are not directly transferable to other environments, a number of studies are needed before general conclusions can be drawn about suitable fault indicators from design. The study reported in this paper provides insight into fault-proneness with respect to a telecommunications system software environment.

Based on the specific environment under investigation, our objective was to study design metrics and their ability to indicate which modules are likely to be fault-prone. A secondary objective is to

identify a suitable methodology combining data collection and statistical prediction model building for use in an industrial environment to identify fault-prone modules. A good design metric for this type of method is one which, with high probability, pinpoints fault-prone modules.

The results reported are based on data obtained from Ericsson Telecom AB, which develops large software intensive systems for telecommunication applications. The data presented was collected from two large software projects. This study illustrates that it is possible to identify a significant number of the fault-prone modules at the design phase. This implies that actions, such as re-design, help in planning verification and validation, resource allocation, and more extensive inspection and testing of specific parts, can be taken early to avoid costly handling of trouble reports from testing and operation.

This paper is organised as follows. In the next section the problem is described more thoroughly. After that a description of the method used to develop and maintain the prediction model is given, emphasising on the applied statistical methods and data collection. In sections 4 and 5, the analysis of release n is presented and the derived model is evaluated. The concluding section summarises the major observations from this case study and discusses implications for other projects

2 Background

The high cost associated with handling faults disclosed during software testing resulted in a number of recent actions to improve service performance at Ericsson. The large number of trouble reports was mainly viewed as a cost problem, but also was viewed as affecting development time and time to market. By applying "quality efforts upstream" (Bergman and Holmquist, 1989) it appeared possible to cut costs for correcting faults, which are generally lower the earlier a fault is found (Bell et al., 1987), and simultaneously shorten development time as less rework would be required.

Since the main part of the total cost of quality deficiency is often found to be caused by very few fault types (Bergman and Klefsjö, 1991), the Pareto principle is often used to identify the most severe problems in a process. The Pareto principle is to concentrate improvement efforts on the *vital few*, not the *trivial many*. There exist a number of examples of the Pareto principle within software engineering. For example, Adams (1984) demonstrated that a small number of faults are responsible for a large number of failures. Munson et al (1992) motivated their discriminative analysis by referring to the 20-80 rule, even though their data demonstrated a 20-65 rule. Zuse (1991) used Pareto techniques to identify the most common types of faults found during function testing. Finally, Shulmeyer et al (1987) described how the principle supports defect identification, inspection and applied statistical techniques.

Fenton et al (1995) presented a four layered approach that supports the establishment of realistic models to address the interrelations between process, product, resources and their attributes. In this study we have investigated the product aspect of one key component within Fenton et al's model: design. As a first step toward a more integrated model, our aim was to investigate the extent to which design complexity can explain the variation in the number of faults. Our intention was also to determine how such models should be established and maintained in the course of evolving software systems and new releases.

The Pareto principle is supported by the data from the projects discussed in this paper, as well as from a previous release of the system. Figure 1 illustrates that 20% of the modules were responsible for approximately 60% of all the trouble reports from testing at Ericsson, which appears to be in line with the data published by Munson et al (1992).

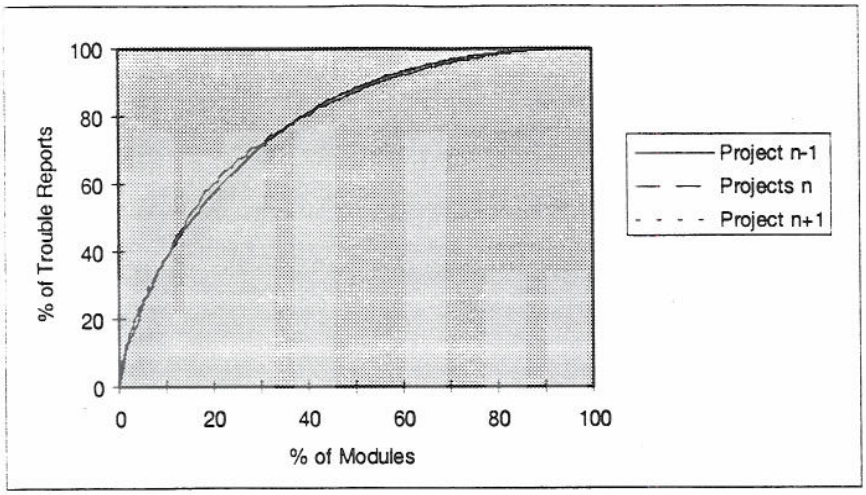

Figure 1: *Pareto diagram from three succeeding projects at Ericsson Telecom AB showing evidence of a 20-60 rule.*

An early identification, at the completion of the design phase, of these vital few modules would enable management to take special measures, such as assigning more experienced developers to support the implementation, additional inspection of design, and more extensive testing. Ultimately, earlier fault detection through identification of the most critical modules means reduced costs, since corrective measures in earlier phases are less expensive. Elimination of faults in later phases also means that time to deliver may in fact be reduced, as less rework is required.

Another objective with this study was to learn which factors influence the number of faults in the software components and at what time this information is available. To accomplish this, methods were developed for measuring probable factors and combining them into prediction models for identifying the most fault-prone modules.

Early identification of the most fault-prone modules has potential benefit in addition to direct cost reduction. For example, knowledge of what causes an error can be used for informing and educating managers, educators, moderators, inspectors, and designers about likely causes of errors. Gained knowledge can also direct process improvement efforts and the development of technical support. This type of learning can help integrate the philosophy of fault avoidance into development phases, so that in future projects fewer faults are created to begin with.

3 Description of the Method

3.1 Motivation for Application of Nonparametric Techniques

The admissible transformations of our empirical data, see table 1, define which statistical methods are relevant (Fenton, 1991). Techniques based on parametric statistics make a number of assumptions that may not be valid. For example Siegel et al (1988) states that a variable must be measured in at least the interval scale to justify the use of parametric methods. Complexity metrics, according to Zuse (1991), are of the ordinal scale.

Scale types	Admissible Transformation	Examples
Nominal	$M' = F(M)$ (F 1-1 mapping)	Labeling/classifying entities
Ordinal	$M' = F(M)$ (F monotonic increasing)	Preference, hardness, air quality
Interval	$M' = \alpha M+\beta \quad (\alpha>0)$	Time, Temperature (Centigrade)
Ratio	$M' = \alpha M \quad (\alpha>0)$	Time interval, length (Absolute)

Table 1: *Admissible transformations for the most important scales (Fenton, 1991).*

The choice of statistical methods is also affected by the research question. Schneidewind (1992) classified models according to three aims: assessment, control and prediction. Depending on the class, Schneidewind suggested different evaluation criteria for evaluating applicability, see table 2. The quality assessment function suggested by Schneidewind provides managers with a rationale basis for assigning priorities for quality improvement effort and allocating personal and computer resources to quality assurance functions. The purpose of the quality control is to identify software, based on predetermined critical values, that has an acceptable quality. The discriminative power reflects the ideas of the Pareto principle, by selecting a threshold that divides the data set into one group with the *vital few* modules (the fault-prone modules) and one group with *trivial many* modules (the non-fault-prone modules). The scale of discriminative power is according to Schneidewind nominal. This is true when the threshold is known and stable over successive projects and releases, which is not always the case.

Quality function	Validation criteria	Scale
Assessment	Association	Interval
	Consistency	Ordinal
Control	Discriminative power	Nominal
	Tracking	Nominal
Prediction	Predictability	Interval, Ratio
All	Repeatability	Ratio

Table 2: *Scales of and thereby applicable statistical methods for the criteria's (Schneidewind, 1992).*

In the analysis of our three projects the Pareto curve appears to be stable, see figure 1. However, the actual threshold for a given percentage varies. Therefore it is not possible to establish a discriminative model for one project or release and transfer it to a succeeding project or release. Even if the threshold was fixed according to the 20-80 rule, the threshold could change from time to time. For example, in one project the management may be prepared to inspect 10% of the modules, and in another management may decide that 20% of the modules would be subject to additional inspection. Modules may also be grouped into smaller discriminative classes (Ebert and Liedtke, 1995) for which different measures should be taken, depending on availability and effectiveness of resources for the different groups. To be able to handle this variation it was desired to build discriminative models based on ordinal reasoning, i.e. the ability of ordering modules should be assessed, so that if the threshold is changed the predictability of the model would remain.

3.2 Measure of Association with Robust Correlation

In this study, we used *Spearman's rank-order correlation coefficient* r_s (Siegel and Castellan, 1988) to predict module rankings with respect to fault-proneness. Spearman's may be the best known nonparametric method. The coefficient measures the correlation between fault-proneness and design, by determining the total magnitude of the discrepancy between the rankings using a simplification of Pearson's product-moment correlation coefficient. The significance of the coefficient can be tested by calculating $z=r_s(N-1)^{1/2}$ and using the table provided by Siegel to determine the probability that the coefficient differs from zero by chance.

Once rankings have been predicted, then the identification of x% of most critical modules can be graphically observed. In this study, a technique similar in concept to Pareto diagrams to assess the accuracy of the prediction was used. The advantage of this technique, called the Alberg diagram (Ohlsson and Alberg, 1994; and Ohlsson, 1993), is that Type I and Type II errors are visible for all thresholds at once. Note that a Type I error means erroneously classifying a fault-prone module as non-fault-prone, and Type II means erroneously classifying non-fault-prone as fault-prone. Normally we are prepared to make some Type II errors if that would help us reduce the number of Type I errors. In the Alberg diagram, the relative increase in the number of modules

needed to be examined to compensate for Type I errors is an indication of the model's predictability. Other techniques, such as the chi-square test, require that a given threshold is specified. In the Alberg diagram, the goodness of a model is expressed by the distance to the optimal curve.

An example-Assume that a system consists of ten modules A to J. The modules are listed in decreasing order with respect to how many trouble reports (*TR*) are written. (See the two first columns in Table 3.) The accumulated number of trouble reports (*Acc TR*) are given on the y-axis in the Alberg diagram, in figure 2.

Modules	TR	Acc TR
A	38	38
B	22	60
C	13	73
D	10	83
E	7	90
F	5	95
G	3	98
H	1	99
I	1	100
J	0	100

Pred1	Acc TR
B	22
C	35
A	73
D	83
E	90
F	95
H	96
G	99
I	100
J	100

Pred2	Acc TR
H	1
E	8
B	30
A	68
C	81
I	82
J	82
G	85
D	95
F	100

Table 3: *The table shows how many trouble reports, TR, each modules has and how two different predictors order the modules. The table also shows the accumulated number of trouble reports.*

If two predictors, *Pred1* and *Pred2*, are to be evaluated, the modules are ordered in decreasing order by the result from each one of the prediction of number of faults. (See the last two columns in Table 3.) By looking at the accumulated number of actual trouble reports, the two predictors can be assessed as seen in figure 2.

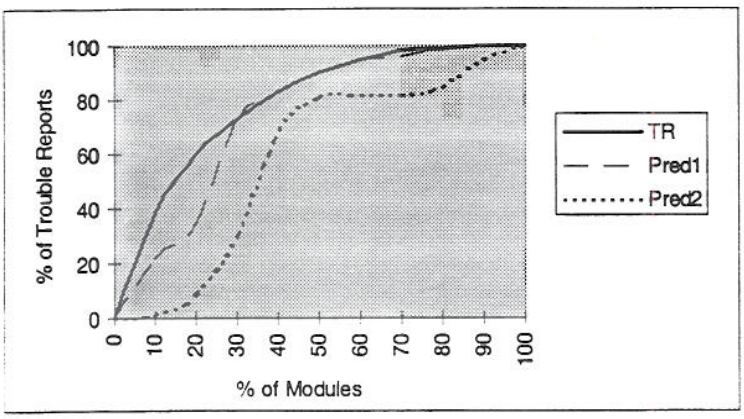

Figure 2: *An example of the Alberg diagram showing the usefulness of two different predictors.*

The line *TR* in figure 2 shows that 20% of all the most fault-prone modules contain 60% of the total number of trouble reports, 50% contain 80% of the total number of trouble reports, and so on. If *Pred1* is used to select 20% of what are assumed to be the most fault-prone modules for which special action should be taken, the result would actually be to find 35% of the faults causing trouble reports. This should be compared with *Pred2* which, for 20% of the modules, would only identify modules that contain 8 percent of the total number of trouble reports. (End example)

Prediction models, equivalent in performance, can be analysed automatically by approximating the area of the difference between the real distribution of TR and each one of the predictors. For a given interval, 0 to n, the distance from every accumulated number of trouble reports to the real

distribution is calculated. The sum of the distances is an approximation of the areas and the smallest area is interpreted as the most useful predictor. The degree of Type I error can be estimated by calculating the necessary increase of a specified threshold in order to identify all fault-prone modules.

3.3 Identification of Predictive Design Metrics

The relationship between various program (or design) metrics and program complexity is a well-studied problem area. (See for example McCabe, 1976; Fenton, 1991; Zuse, 1991; Bache and Bazzana, 1993; and Ross et al., 1993.) One very practical concern in determining which metrics to include in building a prediction model is cost associated with obtaining the actual measurements from a given software system.

There were two important, albeit possibly conflicting, objectives in identifying the set of design metrics to consider. Note that this decision is one which must be made at the pre-data collection stage of model building, before statistical significance tests can be used to screen metrics. One objective is practical: the number must be manageable in size so that data collection does not outweigh the benefit of building the model. The second objective is to identify the "right set", which includes all metrics that can help to make prediction as accurate as possible. Using the existing literature as a guide, experience with the software system under investigation and knowledge about previous characteristics of "trouble areas", an initial set of metrics was selected, see table 4.

Metric	Description
FDL	The number of FDL-objects used.
Dec	Number of decisions.
Con	Number of conditions.
FANin	Number of receive-signals objects.
FANout	Number of send-signals objects.
Sig	Number of signals. Signals are used to communicate with other modules and within a module.
Com	Number of components.
Arc	Number of arcs. Every node is combined with other nodes with help of arcs. Only decisions nodes have a 1 to n connection, all others 1 to 1.
Pat	Number of paths.
Bra	Number of branches (Lennselius, 1986).
Dep	Depth.
Loo	Number of Loops.
Coh	Number of calls to subroutines.

Metric	Description
Mac	Number of subroutines.
MaFDL	Number of FDL-objects in subroutines.
%MaFDL	The percentage of FDL-objects in subroutines, (*MaFDL/FDL*).
MaDec	Number of decisions in subroutines.
%MaDec	The percentage of decisions in subroutines, (*MaDec/Dec*).
MaPat	Number of paths in subroutines.
%MaPat	The percentage of paths in subroutines, (*MaPat/Pat*).
MmFDL	Maximal number of FDL-objects in subroutines.
McC1	McCabe's Cyclomatic complexity (McCabe, 1976).
McC2	Modified McCabe's Cyclomatic complexity, (*Arc-FDL+Com*).
MSDL	Lennselius (Lennselius, 1986) proposed this metric to adjust the complexity, measured by *FDL*, with respect to how many times a similar structure, defined by *FDL, Bra+FANin, and Bra*, appears.
MC(G)	MC(G) is derived by following the same procedure as for MSDL, but use *B+FANin* as the complexity measure.
Ivers	Ivers (Ohlsson, 1993) defines similar structure by *FDL, Pat, Bra*, and *Arc*. The measure is calculated as MSDL.

Table 4: *The design metrics collected for release n.*

Several other issues regarding selection of metrics were considered important, such as the desire to build a predictive model based on observations from one project, to identify cause-and-effect relations as well as with the intention of using it to predict fault-prone modules in the early phases of the **next** project. The latter means that transferability is desirable- i.e. predictor metrics should be robust in the sense that they carry over between successive projects. While significance of a predictor may vary from project to project, in this study it was expected that over time the model would be adjusted to account for new information, while at the same time providing a basis for predicting the immediate next project. Ideally, significance of given metrics should change, since it is meant that the process of studying fault-proneness and identification of cause-and-effect relations results in attention to those factors during development. In particular, it is assumed that

gained knowledge would result in modification of processes and tools to more adequately support the developers during the project. In other words, if our prediction models are not adjusted for changes the prediction models would "wear out".

The combining of metrics into more complex prediction models was directed by finding orthogonality. It is often assumed that different metrics capture different aspects of what is difficult. Still, metrics are often found to be intercorrelated. Principle components have often been used to transform the candidate metrics into orthogonal factors that can be used in place of the original metric (Khoshgoftaar and Lanning, 1994). This results in rather complex models, which naturally are more sensitive to the data set and, as discussed above, more costly to maintain. Therefore our strategy was to study additive and maximum aspects of two and three variables that show evidence of relatively low interrelation, but relatively high correlation to the dependent variable. In the future we also intend to study multiplicative effects.

4 Release n of the switching system

In this section, the process used to collect data and build the initial predictor model is described. The process was based on the experience gained from analysing trouble reports from the nth release of the switching software system. The resulting model is intended for prediction purpose of the (n+1)th release, which will be discussed in section 5.

4.1 Data collection

A number of studies have been reported using metrics extracted from source code, for example (Ebert and Liedtke, 1995; and Munson and Khoshgoftaar, 1992), but few have reported promising prediction results based on design metrics. Khoshgoftaar et al (1996) for example used a subset of metrics that "could be collected from design documentation", but the metrics were extracted from the code. Kitchenham et al (1990) reported on using design metrics, based on Henry and Kafura's information and flow metrics (1981) (1984), for outlier analysis. In our study, direct and indirect measures were calculated by capturing product attributes that were postulated to contribute to the program complexity, see table 4. To make the collection automated, ERIMET (Ohlsson, 1993) was developed. Automation was possible as each module was designed using FCTOOL, a tool for the formal description language FDL which is related to SDL's process diagrams (Turner, 1993). From the FDL-graphs, several direct measures were retrieved.

Communication between modules and within a module are modelled with signals. During the specification phase, the number of new and modified signals (signals are similar to messages) for each module were specified. The count of the number of new and modified signals, *SigFF*, was used as a metric.

The data collected from release n of the switching system was based on 130 modified modules from five subsystems. The size of the modules ranged from 1000 to 6000 LOC. Four of the subsystems were selected as they constituted the base for an earlier project that analysed and classified faults from modules from these four subsystems. The last subsystem was randomly selected. While the method of selecting subsystems could possibly lead to a relatively higher number of fault-prone modules, this appeared not to be the case as the Pareto diagram in figure 1 is quite uniform.

The trouble reports were collected from four different phases: function test (**FT**), system test (**ST**), from the first 26 weeks at a number of site tests (**SI**), and from operation (**OP**). The trouble reports were classified according to whether: (**a**) the fault had already been corrected; (**b**) the

fault will be corrected; (**c**) no action; and (**d**) the fault was due to installation problems. This information is used below to define the dependent variable.

4.2 The Dependent variable

Deriving prediction models for pinpointing the most fault-prone modules assumes that special activities will result, for example more extensive inspection or testing to detect the faults. From the cost aspect, it was therefore necessary to evaluate the modules based on the cost to detect and resolve the trouble reports. The cost of handling trouble reports also depends on the phase in which they were disclosed, as well as the class of trouble report (**a-d** above). Documenting the trouble report (including specifying the fault) was twice as expensive in **OP** as in **FT**. However, the cost for actually correcting a fault increase more rapidly with the phases. For example, it was three time more expensive when a fault was disclosed in **OP** as in **FT**. The correction of failures during **FT** was about 50% more expensive than documenting the trouble report during **FT**. The total cost associated with a module was calculated based on the weights in table 5, and was used as the dependent variable named *costall*.

Phase	Function and System Test		Site test and operation	
Class Action	Documenting	Correcting	Documenting	Correcting
(a) Corrected	1 unit	0	2 units	0
(b) Correct	1 unit	1.6 units	2 units	5 units
(c) No action	1 unit	0	2 units	0
(d) Installation	1 unit	0	2 units	0

Table 5: *The relative cost for handling faults of different types found in different phases.*

Since it could be possible that it is more cost effective to remove 3 faults in two modules, totalling 6 faults, than to remove 5 faults in one module, the dependent variable could be expressed as a density variable, with respect to the detection and removal effort. However, the knowledge about which parameters affect these costs is limited, and until more research has been done in the area it was decided to base the prediction models directly on the number of faults.

It was also important to distinguish between fault-prone and failure-prone modules. The latter reflects a desire to remove faults that have highest impact on the reliability. In doing so two aspects need to be incorporated. The first aspect is the usage profile (Wohlin and Runeson, 1994). Without considering the relative usage of the modules, faults may be removed that never or very seldom manifest themselves as failures (and should therefore be called defects), while the faults that are more likely to manifest as failures are left uncorrected until very late, at worst in operation. The other aspect is concerned with distinguishing between test data and operation data, as well as the classes (**a-d**) of trouble reports. Without knowing if the used test processes disclose real faults or only defects, it is not possible to treat the data as potential failure data. It could also be argued that the faults found during testing are not as interesting as they are already detected before release, and prediction models should therefore be based on operation data. This is not true if the goal is to disclose cause-and-effect relations, nor if the objective is to take special action early, after design and during implementation. With the knowledge at hand the count of all trouble reports classified as b were used as a second dependent variable, named *all-b*.

In analysing the data from an earlier project, it was possible to show a high correlation between *costall* and *all-b*. The Spearman's correlation between *all-b*'s and *costall* for trouble reports was 0.91. This means that it is highly likely that the most fault-prone modules will also be the most costly, even though it was not possible to conclude that most fault-prone modules in test would be the most fault-prone modules in operation. This finding was supported by data from both release n and n+1. Therefore it was determined to only use *all-b* as the dependent variable.

4.3 Prediction Model Based on Release n

As a first step towards the prediction model, Spearman's correlation coefficient was calculated for each candidate metric, using *all-b* as the dependent variable. The objective was to find the most likely metrics that should constitute the prediction model. The correlation varied from 0 to 0.59, and 13 of the variables were in the range of 0.49 to 0.59. The next metric had a correlation of 0.36, apart from *SigFF*, which had a correlation of 0.39. With the same rational used to include *SigFF* in the first case, it was decided to further explore *SigFF* and the 13 metrics with highest correlation.

The next step was to find out if it would be profitable to combine certain variables using additive and maximum functions. The main objective was to find out if any of the candidate variables that showed a strong correlation to the dependent variable and showed low correlation to any other independent variable also showed strong correlation to *all-b* when they were combined. This could be possible as different variables could capture different entities and therefore show a higher correlation to *all-b* when they are combined into one variable or predictor.

If metrics were on a higher scale it would be possible to combine variables with normal mathematical operations. However, these metrics are viewed from a order perspective, and they can not merely be added together, as they differ in range and thus are weighted differently. In this study, equation 1 was used to calculate the additive variable. To calculate the maximum ordering, **mean** was replaced with **max**. In later studies multiplicative effects will be investigated in the same way, as well as different weights for different metrics.

(Eq 1) $\quad add(module_i)=mean(order(met_1(module_i)), order(met_2(module_i)))$

The highest correlation's from combining metrics from the different groups are listed in table 5. Note that correlation values are only slightly higher than for those from the first calculation.

Variables	mean	max
SigFF, Dec	0.5881	0.5910
SigFF, Sig	0.5177	0.5408
SigFF, McC2	0.5925	0.5874
McC2, Sig	0.5961	0.5440
Decision, Sig	0.5690	0.5414
SigFF, McC2, Sig	0.6101	0.5961
SigFF, McC2, FANin	0.6027	0.6042
SigFF, Dec, Sig	0.5891	0.5920

Table 6: *The highest Spearman's correlation using mean and max with two variables and three variables.*

4.4 Evaluation of the Model

The evaluation of the prediction models using the Alberg diagram indicated that using a threshold of 20% would result in the identification of modules responsible for 45% of the faults, and that using a threshold at 30% in 60% of the faults. The best prediction model applicable at the completion of design was *SigFF* and *Dec*. The result can be interpreted as *SigFF* is a measure of the assignment size, and *Dec* as measure of assignment complexity. Adding *Sig* can been seen as incorporating external complexity aspects. However, the observed differences were small. The average strategy was in general better than the maximum.

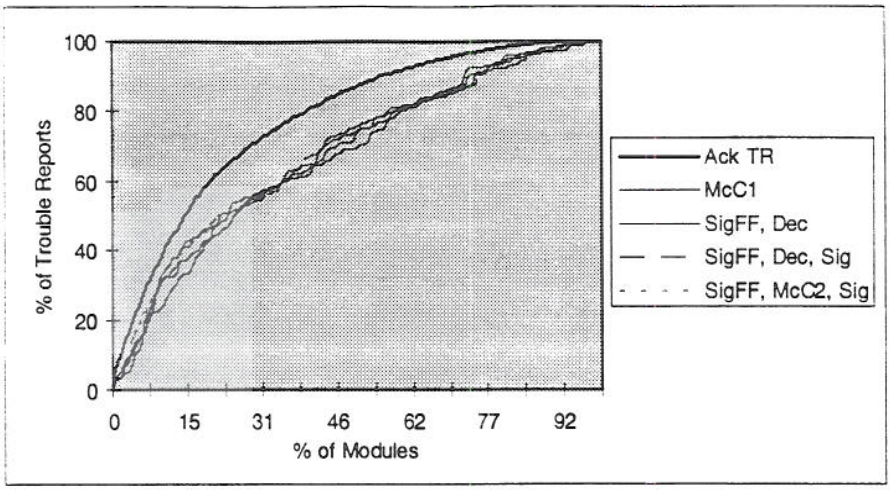

Figure 3: *The Alberg diagram for some of the best prediction models. The prediction models have a distance of 15% at 20%-threshold, 10% at the 40%-threshold.*

5 Validation of the prediction model

5.1 Data collected from release n+1

The data from the subsequent release included more than 230 modules, including 9 modules that had been split into a number of new modules. During the project, 14 new modules were developed which was excluded from the study, as it was assumed that these differed from the reused modules. The module sizes ranged from 1000 to 6000 LOC.

Due to the high cost associated with collecting the large number of different metrics from design only, *Dec* and *Con* were counted as that could be done using UNIX operations without editing the design documents. A cost effective way to collect the number of input-signals (*IS*), as opposed to output-signals, was discovered. As the number of input-signals and output-signals are more or less equal, only the input-signal was counted. *IS* was not the same as *FANin* and *FANout* in release n, since it was not possible with ERIMET to distinguish between those signals that were used for communication with other modules and those used within modules. This focus of signals is motivated by the experience of which types of faults are most often disclosed.

5.2 Applying the prediction model on n+1

The Spearman's correlation coefficient was in general higher for the data of release n+1. The slightly different *FANin*, is increased most to 0.64. The best model was based on *SigFF*, *Dec* and *IS*, and had a correlation of 0.70. The result is displayed in figure 3. If the best prediction model had been applied in the project and 20% used as threshold modules responsible for 49% of the faults, out of 57% possible, would have been identified.

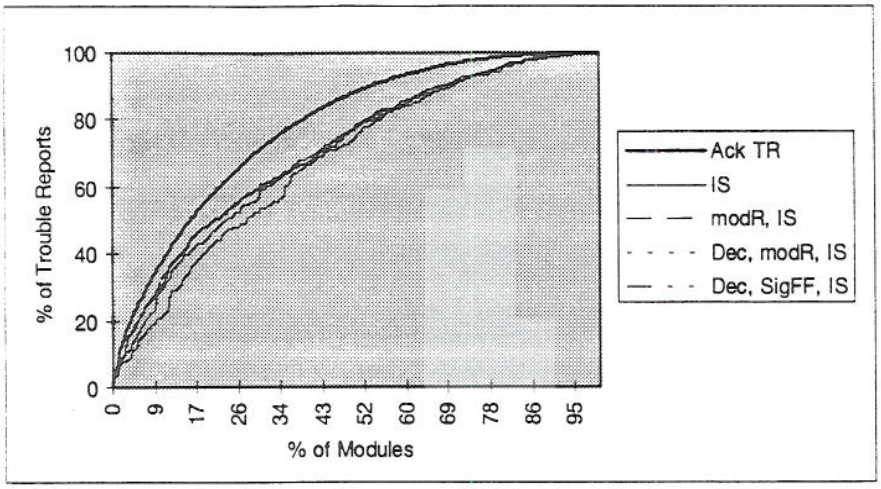

Figure 4: *The Alberg diagram shows the application of the best prediction models derived from release n to the release n+1.*

6 Summary and Conclusions

From the perspective of software development process management, identification of the most fault-prone modules is desirable so that resources aimed at implementation, verification and validation can be allocated in a way that leads to reduced cost. In this paper, a method for predicting the most fault-prone modules by ranking, using Spearman's test to determine the correlation between various design metrics and observed costs of handling trouble reports, is proposed as one possible approach for helping managers to direct effort and subsequently reduce future costs.

One requirement in using this method is that data from past projects are available: in particularly, observations relating costs associated with trouble reports by module are needed (which are the dependent variables) as well as design metrics for each module (which are the independent variables). Ultimately, managers may also want to balance costs associated with resource allocation with the expected benefits from such allocation. Furthermore, the process of reducing costs of handling detected faults is related to the more general goal of attaining software quality in the most cost efficient way. The philosophy behind software quality is to manage processes efficiently (i.e. minimise costs) while at the same time achieve customer requirements, which may be quantified through a measure such as software system reliability.

In our analysis of three successive releases of a switching system software product, the Pareto principle was confirmed. The data from three projects support the Pareto principle, and states that 20% of the modules causes about 60% of the trouble reports. The analysis reported in this paper indicate that it should be possible to identify the 20% the modules causing 50% of the faults. It is believed that the result could only be improved slightly, unless one attempts to incorporate process and resources attributes as suggested by Fenton et al (1995).

Projects that adopt the strategy for building prediction models to improve project control will be required to enlarge their data collection to incorporate data about resource (efforts) and inspection (number of faults detected). Without such information, prediction models can not be maintained or evaluated. Finally, while the parameters used in this study may suggest guidance for other similar applications, it is felt that the strategy and not the particular choice of prediction model is the most important aspect learned from this experience. The transferability of the outlined strategy need to be further evaluated by applying it to other telecommunication application and other specific environments.

Acknowledgements

The authors would like to thank Norman Fenton and colleagues at Ericsson Telecom AB for supporting our work. This work was supported by the Swedish Institute for applied mathematics (ITM).

References

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

Bache, R. and Bazzana, G. (1993). *Software Metrics for Product Assessment*. McGraw-Hill.

Bell, D., Morrey, I., and Pugh, J. (1987). *Software Engineering: A Programming Approach*. Prentice-Hall.

Bergman, B. and Holmquist, L. (1989). *A Swedish programme on robust design and Taguchi methods*. In Taguchi Methods-Proceedings of the 1988 European Conference, pages 189-202. Elsevier Applied Science.

Bergman, B. and Klefsjö, B. (1991). *Kvalitet - från behov till användning*. Studentlitteratur.

Boehm, B. W. (1981). *Software engineering economics*. Prentice-Hall.

Ebert, C. and Liedtke, T. (1995). *An integrated approach to criticality prediction*. In The Sixth International Symposium on Software Reliability Engineering, pages 14-23.

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.

Fenton, N. E. (1991). *Software metrics-a rigorous approach*. Chapman & Hall.

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

Henry, S. and Kafura, D. (1984). *The evaluation of software system's structure using quantitative software metrics*. Software-Practise and Experience, 14(6):5561-573.

Khoshgoftaar, T. M. and Lanning, D. L. (1994). *Are the principal components of software complexity data stable across software products?* In Second International Software Metrics Symposium, pages 61-72.

Khoshgoftaar, T. M., Allen, E. B., Kalaichelvan, K. S., and Goel, N. (1996). *Early quality prediction: a case study in telecommunications*. IEEE Software, 13(1):65-71.

Kitchenham, B. A., Pickard, L. M., and Linkman, S. J. (1990). *An evaluation of some design metrics*. Software Engineering Journal, 5(1):50-58.

Lennselius, B. (1986). *Software complexity and its impact on different software handling processes*. In 6th International Conference on Software Engineering for Telecommunications Switching Systems, pages 148-153.

Lennselius, B. (1990). *Estimation of software fault content for telecommunication systems*. Technical Report 104, Department of Communication systems, Lunds Institute of Technology.

McCabe, T. J. (1976). *A complexity measure*. IEEE Transaction on Software Engineering, 2(4):308-320.

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

Ohlsson, N. (1993). *Predicting error-prone software modules in telephone switches*. Master's thesis, Department of computer and information science, Linköping University.

Ohlsson, N. and Alberg, H. (1994). *Predicting fault-prone software modules in telephone switches*. Submitted to IEEE Transaction on Software Engineering.

Ross, M., Brebbia, C. A., Staples, G., and Stapleton, J., editors (1993). *Software Quality Management*, chapter The automation of software process and product quality, pages 727-744. Elsevier.

Schneidewind, N. F. (1992). *Methodology for validating software metrics*. IEEE Transaction on Software Engineering, 18(5):410-422.

Schulmeyer, G. G. and McManus, J. I., editors (1987). *Handbook of software quality assurance*. van Nostrand Reinhold Company.

Siegel, S. and Castellan, N. J. (1988). *Nonparametric statistics for the behavioral sciences*. McGraw-Hill, 2nd edition.

Turner, K. J., editor (1993). *Using formal description techniques - An introduction to ESTELLE, LOTOS and SDL*. John Wiley & Sons.

Wohlin, C. and Runeson, P. (1994). *Certification of software components*. IEEE Transaction on Software Engineering, 20(6):494-499.

Zuse, H. (1991). *Software complexity-measures and methods*. Walter de Gruyter.