DESIGN AND DEVELOPMENT OF SOFTWARE FAULT PREDICTION MODEL TO ENHANCE THE SOFTWARE QUALITY LEVEL

Dr. S. Ravichandran
CEO & Chief Scientist, Trimentus Technologies
T. Nagar, Chennai, India
Email: drravis@trimentus.com

ABSTRACT

Software fault prediction models play an important role in software quality assurance. They identify software subsystems (modules, components, classes, or files) which are likely to contain faults. These subsystems, in turn, receive additional resources for verification and validation activities. Fault prediction models are binary classifiers typically developed using one of the supervised learning techniques from either a subset of the fault data from the current project or from a similar past project. In practice, it is critical that such models provide a reliable prediction performance on the data not used in training. Variance is an important reliability indicator of software fault prediction models. Models were built using both, original software metrics (RAW) and their principle components (PCA). Two-way ANOVA randomized-complete block design models with two blocking variables are designed with average absolute and average relative errors as response variables. System release and the model type (RAW or PCA) form the blocking variables and the prediction technique is treated as a factor. Using multiple-pairwise comparisons, the performance order of prediction models is determined. We observe that for both average absolute and average relative errors, the CART-LAD model performs the best while the S-PLUS model is ranked sixth. Software testing is a crucial activity during software development and fault prediction models assist practitioners herein by drawing upon the machine learning literature. While especially the Naive Bayes classifier is often applied in this regard, citing predictive performance and comprehensibility as its major strengths, a number of alternative Bayesian algorithms that boost the possibility of constructing simpler networks with fewer nodes and arcs remain unexplored.
1. INTRODUCTION
Software reliability is an important attribute of high-assurance and mission-critical systems. Such complex systems are heavily dependent on reliability and stability of their underlying software applications. The challenges involved in achieving high software reliability increases the importance in developing and quantifying measures for software quality. Early fault prediction is a proven technique in achieving high software reliability, and can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults. A software fault is a defect that causes software failure in an executable product.

2. FAULT PREDICTION MODELLING
Fault prediction modelling has become a popular method for the early identification of fault-prone code. It is now the topic of a large body of software engineering literature. In this section we discuss factors which underpin the development of a prediction model likely and present basic recommendations relating to principles of measuring the performance of such models.

2.1 Developing a fault prediction model
Developing a reliable fault prediction model requires a number of model building concepts to be appropriately addressed. These concepts must be taken into account when reviewing models of fault prediction. These essential model building concepts include:

1. Predictor or independent variables. These are usually metrics based on software artefacts, such as static code metrics or change data, and are usually features that enable some degree of fault prediction. This review is scoped to those models using independent variables based on units of code, such as files, classes or modules of code.

2. Output or dependent variables. The output from the model is a prediction of fault proneness in terms of faulty versus non-faulty code units. This output typically takes the form of either categorical or continuous output variables. Categorical outputs classify code units as either faulty or non-faulty. Continuous outputs usually provide the number of faults in a code unit3.

3. Modelling methods. One or more modelling methods are used to explore the relationship between the predictor variables (or independent variables) and the outputs (or dependent variables). These methods may be, for example, types of statistic like regression or machine learning.

2.2 Measuring the performance of prediction models
Measuring the predictive performance of models is an essential part of model development and subject to on-going debate in the literature.

Reporting performance is often based on the analysis of data in a confusion matrix as shown in Table 1 and explained further in Table 2. This matrix reports how the model classified the different fault categories compared to their actual classification (predicted versus observed). Many performance measures are related to components of the confusion matrix shown in Table 2. These components can be either within a confusion matrix or used individually. Confusion matrix measures of performance are most relevant to fault prediction models producing categorical outputs, though
continuous outputs can be converted to categorical outputs and analysed in terms of a confusion matrix.

**Table 1** A confusion matrix

<table>
<thead>
<tr>
<th>Observed defective</th>
<th>Predicted defective</th>
<th>Predicted defect free</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
<td></td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2** Confusion matrix based performance indicators

<table>
<thead>
<tr>
<th>Construct</th>
<th>Also known as</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Positive</td>
<td>FP, and Type I Error</td>
<td>Classifies non faulty unit as faulty</td>
</tr>
<tr>
<td>False Negative</td>
<td>FN, and Type II Error</td>
<td>Classifies faulty unit as not faulty</td>
</tr>
<tr>
<td>True Positive</td>
<td>TP</td>
<td>Correctly classified as faulty</td>
</tr>
<tr>
<td>True Negative</td>
<td>TN</td>
<td>Correctly classified as non faulty</td>
</tr>
</tbody>
</table>

### 3. PROPOSED MODEL

Prediction of faults is desirable for any industry and attracts both engineers as well as managements. For software industry it provides an opportunity for the early identification of software quality, cost overrun and optimal development strategies. During earlier phases of software development; predicting the number of faults can reduce the efforts for additional reviews and more extensive testing.

The model architecture is shown in Figure1. Stages present in the proposed structure are similar to the waterfall model, a well known software development process model. It divides the structure into four consecutive phase I, II, III, and IV i.e. requirement, design, coding, and testing phase respectively. Phase-I predicts the fault density at the end of requirement phase using relevant requirement metrics. Phase-II predicts the fault density at the end of design phase using design metrics as well as output of the requirements phase. Similarly at phase-III besides the using coding metrics, output of phase-II is also considered as input to predict the fault density at the end of coding phase. Finally, the phase-IV predicts the fault density using testing metrics as well as output of the coding phase.
4. EXPERIMENTAL DESIGN IN EMBEDDED SYSTEMS

From an industrial perspective, software managers aim to decrease their testing efforts while decreasing fault rates, thereby producing high quality real time systems. We observed that cluster based classifiers would detect 76% of the faulty modules with precision of 68%. Also the cluster based classification decreases the false alarms rates. In commercial applications, companies need to employ cost effective oracles, since an increase in false alarms would waste inspection costs by guiding testers through actually safe modules.

Discretisation is the transformation of a continuous variable into a discrete space, grouping together multiple values of a continuous attribute, and partitioning the continuous domain into a finite numbers of nonoverlapping intervals. The task of discretizing an input attribute for classification problems is usually divided into supervised discretisation, when knowledge interdependency between the class level and attribute values is used for the discretization process and unsupervised discretization, when the class values of the instances are unknown or not used. The methods for unsupervised discretization are equal-width and equal-frequency binning. The equal width divides the range of values of a numerical attribute into a predetermined number of equal intervals. The equal frequency divides the range of values into a pre-determined number of intervals that contain equal number of instances.
5. FAULT PREDICTIONS IN OBJECT ORIENTED SOFTWARE

Our module allows the user to input a folder which contains the classes to be tested. On loading the folder the selected classes to be tested present inside the folder are fed to the source code one by one which monitors for the occurrence of error. The errors are monitored based on the six metrics specified earlier. If an error is detected then the error number is stored in the error list of that particular class in the database.

6. ADVANTAGES OF PREDICTION MODELS

1) The model itself is just an indicator and should not be trusted entirely. It is good practice to compare the models output with engineer expectations. Usually, you see a large overlap between both risk lists. The difference between both lists is the interesting parts--areas that the model considers risky while the engineers do not have this area on the radar. These might be the areas of highest risk as nobody cared about them. This can be true for both static waterfall, agile, and intermediate processes.

2) The model should be tightly bound against the development period and you should ensure that already fixed bugs are recognized. Most prediction models published do not consider continuous usage of these models and are based on measurements that cannot capture already applied bug fixes. For example, consider a prediction model based on code dependencies. Now assume, you hunt all the bugs in all the files predicted most defect-prone and you run the model again. Well, you will get the same result from the model, because fixing a bug will not change code dependencies significantly. At this point, your metric is gone since you deliberately destroyed the correlation between dependencies and defects by fixing the defects. Such a model will not work in an agile setting. There it is important to choose dynamic and self-adapting measurements that capture already fixed bugs. An example might be change bursts, test metrics (e.g. increase in coverage, number of new scenario tests), and in general measurements that capture the difference between two revisions.

7. CONCLUSIONS

This review shows that many fault prediction models have been published in the last ten years. These models are heterogeneous and come in all shapes and sizes. A wide range of statistical and machine learning approaches are used to build models. An encouragingly high proportion of studies have been developed using industrial data, with OSS data also featuring strongly. Many studies use the publicly available NASA data. A wide variety of independent variables have been used in models, the most common of which fall into categories such as static code metrics, change metrics and previous fault metrics. However, there is no clear ‘best’ indicator of fault proneness emerging from studies, with indicators performing differently across different studies. Independent variables such as lines of code, complexity metrics, process metrics, module size, age of file were strongly correlated to faults in some studies but had no correlation to faults in others. This suggests that there is no single ‘best’ approach to predicting faults across all problem domains. The challenge remains to identify the context variables that determine a model’s applicability.
REFERENCES


AUTHOR PROFILE

**Dr S. Ravichandran** is a Software Quality Professional. He is also a Six Sigma Master Black Belt with academic qualifications ME, MBA, Ph.D. He has carried out software quality appraisals, consultancy and training services in India and over 20 countries such as China, Brazil, Singapore, Malaysia, US, Europe, Middle East etc. He is presently the Chairman, CEO and Chief Scientist of Trimentus Technologies, Chennai & US. He is also well known expert in engineering quality including NDT techniques.