Editor: **Tore Dybå**
SINTEF
tore.dyba@sintef.no

Editor: **Helen Sharp**
The Open University, London
h.c.sharp@open.ac.uk

Developing Fault-Prediction Models: What the Research Can Show Industry

Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell

CODE FAULTS ARE a daily reality for software development companies. Finding and removing them costs the industry billions of dollars each year. The lure of potential cost savings and quality improvements has motivated considerable research on fault-prediction models, which try to identify areas of code where faults are most likely to lurk. Developers can then focus their testing efforts on these areas. Effectively focused testing should reduce the overall cost of finding faults, eliminate them earlier, and improve the delivered system's quality. Problem solved.

Analysis of 206 fault-prediction models reported in 19 mature research studies reveals key features to help industry developers build models suitable to their specific contexts.

So why do so few companies seem to be developing fault-prediction models?

One reason is probably the sheer number and complexity of studies in this field. Before companies can start to develop the models, they must understand questions such as which metrics to include, which modeling techniques perform best, and how context affects fault prediction.

To answer these questions, we conducted a systematic review of the published studies of fault prediction in code from the beginning of 2000 to the end of 2010.[1] On the basis of this review and subsequent analysis of 206 models, we present key features of successful models here.

Context Is Key

We found 208 studies published on predicting code faults over the 11-year period covered in our review. These studies contain many hundreds of individual models, whose construction varies considerably.

Although it would be nice if companies could simply select one of these published models and use it in their own environment, the evidence suggests that fault-prediction models perform less well when transferred to different contexts.[2] This means that practitioners must understand how existing models might or might not relate to their own model development. In particular, they must understand the specific context in which an existing model was developed, so they can base their own models on those that are contextually compatible.

However, we found three challenges to this apparently simple requirement. First, many studies presented insufficient information about the development context. Second, most studies reported models built using open source data, which can limit their compatibility with commercial systems. Third, it remains unclear which context variables (application domain, programming language, size, system maturity, and so on) are tied to a fault-prediction model's performance.

Establishing Confidence in Existing Models

Practitioners need a basic level of confidence in an existing model's performance. Such confidence is based on

0740-7459/11/$26.00 © 2011 IEEE

understanding how well the model has been constructed, its development context, and its performance relative to other models.

Our review suggests that few of the published models provide sufficient information to adequately support this understanding. We developed a set of criteria based on research (for example, see Kai Petersen and Claes Wohlin[3]), to assess whether a fault-prediction study reports the basic information to support confidence in a model. Figure 1 shows a checklist based on these criteria (details are available elsewhere [1]).

When we applied these criteria to the 208 studies we reviewed, only 36 passed all criteria. Each of these 36 studies presents clear models and provides the information necessary to understand the model's relevance to a particular context.

We quantitatively analyzed the performance of the models presented in 19 of the 36 studies. These 19 studies all report categorical predictions, such as whether a code unit was likely to be faulty or not (see the sidebar). Such predictions use performance measures that usually stem from a confusion matrix (see Figure 2). This matrix supports a performance comparison across categorical studies.

We omitted 13 studies from further analysis because they reported continuous predictions, such as how many faults are likely to occur in each code unit. Such studies employ a wide variety of performance measures that are difficult to compare. Likewise, we omitted four categorical studies that reported performance data based on the area under a curve.

The 19 studies reporting categorical predictions contained 206 individual models. For each model, we extracted performance data for

- *precision* = TP/(TP + FP): proportion of units predicted as faulty that were faulty;
- *recall* = TP/(TP + FN): proportion of faulty units correctly classified; and
- *f-measure* = (2 × recall × precision)/(recall + precision): the harmonic mean of precision and recall.

For studies that didn't report these measures, we recomputed them from the available confusion-matrix-based data. This let us compare the performance of all 206 models across the 19 studies and to draw quantitative conclusions.

Phase One: Prediction Criteria
Is a prediction model reported?
Is the prediction model tested on unseen data?
Phase Two: Context Criteria
Is the source of data reported?
Is the maturity of data reported?
Is the size of data reported?
Is the application domain of data reported?
Is the programming language of data reported?
Phase Three: Model Criteria
Are the independent variables clearly reported?
Is the dependent variable clearly reported?
Is the granularity of the dependent variable reported?
Is the modeling technique used reported?
Phase Four: Data Criteria
Is the fault data acquisition process described?
Is the independent variables data acquisition process described?
Is the faulty/nonfaulty balance of data reported?

FIGURE 1. Checklist of criteria for establishing confidence in a model. Only 36 of 208 studies from the systematic literature review met all criteria.

	Number of code units	
	Predicted faulty	Predicted not faulty
Actually faulty	True positive (TP)	False negative (FN)
Actually not faulty	False positive (FP)	True negative (TN)

FIGURE 2. Confusion matrix. The two columns and two rows capture the ways in which a prediction can be correct or incorrect.

FAULT-PREDICTION MODEL ELEMENTS

There are three essential elements to a fault-prediction model.

PREDICTOR VARIABLES

These independent variables are usually metrics based on software artifacts, such as static code or change data. Models have used a variety of metrics—from simple LOC metrics to complex combinations of static-code features, previous fault data, and information about developers.

OUTPUT VARIABLES

A model's output, or independent variable, is a prediction of fault proneness in terms of faulty versus nonfaulty code units. This output typically takes the form of either *categorical* or *continuous* output variables.

Categorical outputs predict code units as either faulty or nonfaulty. Continuous outputs usually predict the number of faults in a code unit. Predictions can address varying units of code—from high-granularity units, such as plug-in level, to low-granularity units, such as method level.

MODELING TECHNIQUES

Model developers can use one or more techniques to explore the relationship between the predictor (or independent) variables and the output (or dependent) variables. Among the many available techniques are statistically based regression techniques and machine-learning techniques such as support vector machines. Ian Witten and Eibe Frank provide an excellent guide to using machine-learning techniques.[1]

Reference

1. I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2005.

What Works in Existing Models

When compared with each other, most of the 206 models peaked at about 70 percent recall—that is, they correctly predict about 70 percent of actual real faults. Some models performed considerably higher (for example, see Shivkumar Shivaji and his colleagues[4]), while others performed considerably lower (see Erik Arishholm and his colleagues[5]).

Models based on techniques such as naïve Bayes and logistic regression seemed to perform best. Such techniques are comparatively easy to understand and simple to use.

Additionally, the models that performed relatively well tended to combine a wide range of metrics, typically including metrics based on the source code, change data, and data about developers (see Christian Bird and his colleagues[6]). Often, the models performing best had optimized this set of metrics (for example, by using Principal Component Analysis or Feature Selection as in Shivaji[4]). Models using source-code text directly as a predictor yielded promising results (see Osamu Mizuno and Tohru Kikuno[7]).Models using static-code or change-based metrics alone performed least well. Models using LOC metrics performed surprisingly competitively.

Our systematic literature review suggests first that successful fault-prediction models are built or optimized to specific contexts. The 36 mature studies we identified can support this task with clearly defined models that include development contexts and methodologies. Our quantitative analysis of the 19 categorical studies from this set further suggests that successful models are based on both simple modeling techniques and a wide combination of metrics. Practitioners can use these results in developing their own fault-prediction models.

Acknowledgments

We thank the UK's Engineering and Physical Science Research Council, which supported this research at Brunel University under grant EPSRC EP/E063039/1, and the Science Foundation Ireland, which partially supported this work at Lero under grant 3/CE2/I303_1. We are also grateful to Sue Black and Paul Wernick for providing input to the early stages of the work reported.

References

1. T. Hall et al., "A Systematic Review of Fault Prediction Performance in Software Engineering," accepted for publication in *IEEE Trans. Software Eng.*; preprint available at http://bura.brunel.ac.uk/handle/2438/5743.
2. N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," *Proc. 28th Int'l Conf. Software Eng.*, (ICSE 06), ACM Press, 2006, pp. 452–461.
3. K. Petersen and C. Wohlin, "Context in Industrial Software Eng. Research," *Proc. 3rd Int'l Symp. Empirical Software Eng. and Measurement* (ESEM 09), IEEE CS Press, 2009, pp. 401–404.
4. S. Shivaji et al., "Reducing Features to Improve Bug Prediction," *Proc. 24th IEEE/ACM Int'l Conf. Automated Software Eng.* (ASE 09), IEEE CS Press, 2009, pp. 600–604.
5. E. Arisholm, L.C. Briand, and E.B. Johannessen, "A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models," *J. Systems and Software*, vol. 83, no. 1, 2010, pp. 2–17.
6. C. Bird et al., "Putting It All Together: Using Socio-Technical Networks to Predict

Failures," *Proc. 20th Int'l Symp. Software Reliability Eng.* (ISSRE 09), IEEE CS Press, 2009, pp. 109–119.

7. O. Mizuno and T. Kikuno, "Training on Errors Experiment to Detect Fault-Prone Software Modules by Spam Filter," *Proc. 6th Joint Meeting European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng.* (ESEC-FSE 07), ACM Press, 2007, pp. 405–414.

TRACY HALL is a reader in software engineering at Brunel University, UK. Contact her at tracy.hall@brunel.ac.uk.

SARAH BEECHAM is a research fellow at Lero, the Irish Software Engineering Research Centre. Contact her at sarah.beecham@lero.ie.

DAVID BOWES is a senior lecturer at the University of Hertfordshire, UK. Contact him at d.h.bowes@herts.ac.uk.

DAVID GRAY is a PhD student at the University of Hertfordshire, UK. Contact him at d.gray@herts.ac.uk.

STEVE COUNSELL is a reader in software engineering at Brunel University, UK. Contact him at steve.counsell@brunel.ac.uk.