FILIPPO LANUBILE, UNIVERSITY OF MARYLAND

Why Software Reliability Predictions Fail

New views of mature ideas on software and quality productivity.

Software reliability reflects a customer's view of the products we build and test, as it is usually measured in terms of failures experienced during regular system use. But our testing strategy is often based on early product measures, since we cannot measure failures until the software is placed in the field. This issue, Filippo Lanubile shows us that such measurement is not effective at predicting the likely reliability of the delivered software.

—Shari Lawrence Pfleeger

SOFTWARE ENGINEERS HAVE A GREAT interest in applying measurement for predictive purposes. Most such studies focus on predicting early in the life cycle the reliability of software components. Because accurate prediction allows extra effort to be dedicated to inspecting and testing fault-prone components, its expected benefit is a more reliable product at a lower cost. Many predictive techniques are available, however. Discerning the real effectiveness of a particular technique, given the sometimes contradictory results presented by its advocates, requires further study.

PREDICTABLE PATTERNS. My survey of the literature for constructing reliability prediction systems shows the following patterns:

- Predictor variables are often product metrics, usually measuring design and code characteristics, and sometimes documentation attributes.
- No unique set of product metrics is used across all the studies, even for those performed by the same authors at different times.
- Direct measures of reliability differ among the studies; many measure the number of faults discovered during testing, some count the number of failures during operation, and others track the number of repairs made during maintenance.
- Prediction problems are often reduced to classification problems by choosing a categorical variable as an outcome that may be predicted from the predictor variables, for example, grouping all components with at least one fault under the high-risk category and all components with no faults under the low-risk category.
- Many study authors advocate a modeling technique derived from statistical analysis, machine learning, or neural networks—or a combination of the three—aiming to show that their own approach is "good," without comparing it to any others.
- Other study authors show that their modeling technique is "better than others," by comparing it with just one or two different techniques selected to showcase their own technique's advantages.
- Studies use different criteria when comparing various prediction systems, use different definitions even when using the same criteria names, risk ambiguity when they use informal criteria, and fail to capture all aspects of the prediction systems studied.

Few studies define criteria so that the capability to actually predict behavior can be determined.

- Few studies try to define criteria so that the capability of the model to actually predict real behavior can be determined.
- All the studies claim to have been successful at showing the superiority of the advocated modeling technique; when studies include a comparison, the advocated technique always scores highest.

EMPIRICAL STUDY. Amazed by previous studies' high success rate when using predictive techniques, but conscious of existing methodological limits, I started a research project with Giuseppe Visaggio to externally replicate past studies. Scientists perform replications to increase their ability to generalize their results in different settings and times. External replications—those independently conducted by different researchers—are needed because empirical observations in support of a hypothesis may be in error or be biased by the original researcher. Our replication effort exhibited the following characteristics.

We used our own data, collected during three

Editor:
Shari Lawrence Pfleeger
Systems/Software
4519 Davenport St. NW
Washington, DC
20016-4415
s.pfleeger@ieee.org

0740-7459/96/$05.00 © 1996 IEEE

TABLE 1
RESULTS FROM COMPARING REAL AND PREDICTED RISK

Modeling technique	Predictive Validity (Probability)	Proportion of Type 1 Errors	Proportion of Type 2 Errors	Proportion of Type 1 + 2 Errors	Completeness	Overall Inspection	Wasted Inspection
Discriminant analysis	p=0.621	28%	26%	54%	42%	46%	56%
Principal-component analysis + discriminant analysis	p=0.408	15%	41%	56%	68%	74%	55%
Logistic regression	p=0.491	28%	28%	56%	42%	49%	58%
Principal component analysis + logistic regression	p=0.184	13%	46%	59%	74%	82%	56%
Logical classification Model	p=0.643	26%	21%	46%	47%	44%	47%
Layered neural network	p=0.421	28%	28%	56%	42%	49%	58%
Holographic network	p=0.634	26%	28%	54%	47%	51%	55%
Heads or tails?	p=1.000	25%	25%	50%	50%	50%	50%

years of a project-intensive software engineering course held at the University of Bari. Small teams of students developed 27 Pascal programs in the IS domain from the same specification. Other independent student teams drawn from an advanced software engineering course tested different groups of components, randomly selected from each program.

We chose the most-used dependent and independent variables. The dependent variable was the risk class of software components. We defined as high-risk any software component whose faults were detected during testing, and as low-risk any component with no discovered faults. The two classes contained an approximately equal number of components. The independent variables were 11 product metrics covering design attributes such as fan-in, fan-out, and information flow; implementation attributes such as cyclomatic complexity, number of unique operands, total number of operands, total lines of code, number of noncomment lines of code, and Halstead's program length and volume; and the documentation attribute of comment density.

We included all the classification techniques already used for predicting software reliability: principal-component analysis, discriminant analysis, logistic regression, logical classification models, layered neural networks, and holographic networks. Principal-component analysis served as a preprocessing step to obtain a smaller number of orthogonal domain metrics. We built two models each for discriminant analysis and logistic regression. The first model was based on the 11 original complexity measures; the second used three domain metrics that had been generated from the principal-component analysis.

We improved the evaluation criteria. All criteria were formalized using a two-way contingency table as the underlying model with one row for each level of the variable real risk and one column for each level of the variable predicted risk. We assessed the absolute worth of a predictive model by testing a null hypothesis of no association between the real risk and the predicted risk with an alternative hypothesis of general association.

Among the criteria to compare the performance of the prediction systems, we measured the proportions of the two misclassifications: Type 1 errors, in which a high-risk component is classified as low-risk, and Type 2 errors, in which a low-risk component is classified as high-risk. We measured the completeness of the predic-

Continued on page 137

point. My favorite tidbit describes approximate pattern matching with or without errors, with "don't care symbols," and so on. It is based on the concept of editdistance, which tells you how close two strings are to each other in terms of elementary operations on characters such as change, insert, and delete. This is great fun and applications rush to mind. Allow me some nit-picking, though. To implement "permissive" command-line interfaces, our company once used Levenshtein distance or Levenshtein matching, which seems similar to the editdistance concept. Since the documentation has long since vanished, I asked our Internet genie about Levenshtein. It came back with a single useful citation—from Professor Crochemore's institute. The abstract, unfortunately, didn't help much. So, what *is* the difference? An answer, as well as pointers to the relevant literature, would be welcome now that helpful query systems such as wizards and intelligent agents have become fashionable.

The chapter on compression, on the other hand, offers nothing new. It chiefly consists of a classical, if terse, discussion of Huffman coding and its generalizations and of the Ziv-Lempel algorithm. Because the book deals with text, only lossless methods are considered. This could explain the chapter's brevity. The numerous references to the literature should allow you to dig deeper if needed. This, by the way, also applies to the rest of the book. For digging even deeper still, connect to the WWW site of Institute Gaspard Monge, University of Marne la Vallée, where Professor Maxime Crochemore is working (htpp://monge.univ-mlv.fr/ resumes94.html).

Despite my need to enter this book one vanishingly small step at a time, it impressed me. Its audience might be limited, but the appeal of its material should not be. May its algorithms percolate down to levels where they will meet with rewarding implementations that, I hope, don't lose too much of their severe elegance. ◆

Text Algorithms by Maxime Crochemore and Wojciech Rytter, Oxford University Press, Inc., New York, 1994, 412 pp., $95.

READER SERVICE NUMBER 54

quality time

Continued from page 132

tion, defined as the percentage of high-risk components that actually have been classified as such by the model. We also considered the cost of identifying a component as needing more verification effort. We measured both the overall inspection cost, defined as the percentage of components that have been flagged as high-risk, and the wasted inspection, defined as the percentage of verified components that have been incorrectly classified.

RESULTS. We built a training set, including two-thirds of the 118 tested components, to create and tune the predictive models. We used the remaining third of the components, which comprised the testing set, to assess the models and compare their performances, as shown in Table 1. Despite the variegated selection of techniques available, no model satisfied the criterion of predictive validity by being able to discriminate between components with faults and components without faults. All the significance probabilities are too high with respect to the most common values—0.01, 0.05, 0.1—used to reject a null hypothesis. No model shows a significant departure from the performance of a random prediction, except for discriminant analysis and logistic regression applied in conjunction with the principal-component analysis. These two models have good percentages of Type 1 error and completeness but also bad percentages of Type 2 error and overall inspection. The proportion of Type 1 + Type 2 errors and the wasted inspection do not vary with respect to the other models. Instead of producing more stable models, principal-component analysis built models that were biased toward classifying components as high-risk.

LESSONS LEARNED. Our experience indicates that the future behavior of software products cannot always be predicted successfully. Although this sounds obvious, much of the scientific literature reports successful results only in the identification of fault-prone components from product measures. Publishing only empirical studies with positive findings can give practitioners unrealistic expectations that are quickly followed by equally unrealistic disillusionment.

Although the study did not specifically set out to do so, it shows that predictive-modeling techniques are only as good as the data on which they are based. All the prediction systems failed because they assumed a relationship between software product measures and software faults. This is not always true. On the contrary, a predictive model, from the simplest to the most complex, is worthwhile only if used with a local process to select metrics that are valid as predictors. Recently, methods to validate measures of internal software attributes have been proposed, based on iterative verification of locally collected data. Unfortunately, these methods cannot guarantee *a priori* that a significant relationship will be found between some internal product metric and the software attribute of interest. ◆

Filippo Lanubile is a senior researcher of the Experimental Software Engineering Group at the University of Maryland, College Park (http://www.cs.umd.edu/projects/SoftEng/ESEG/). He is currently on sabbatical from the University of Bari, Italy, where he is an assistant professor of computer science. He can be reached at lanubile@cs.umd.edu.