Information and Software Technology 000 (2017) 1–27

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Risk-averse slope-based thresholds: Definition and empirical evaluation

Sandro Morasca*, Luigi Lavazza

Università degli Studi dell'Insubria, Department of Theoretical and Applied Sciences, Italy

ARTICLE INFO

Article history:
Received 19 September 2016
Revised 4 March 2017
Accepted 16 March 2017
Available online xxx

Keywords:
Fault-proneness
Faultiness
Threshold
Software measures
Logistic regression
Probit regression
Risk-aversion

ABSTRACT

Background. Practical use of a measure X for an internal attribute (e.g., size, complexity, cohesion, coupling) of software modules often requires setting a threshold on X, to make decisions as to which modules may be estimated to be potentially faulty. To keep quality under control, practitioners may want to set a threshold on X to identify "early symptoms" of possible faultiness of those modules that should be closely monitored and possibly modified.

Objective. We propose and evaluate a risk-averse approach to setting thresholds on X based on properties of the slope of statistically significant fault-proneness models, to identify "early symptoms" of module faultiness.

Method. To this end, we introduce four ways for setting thresholds on X. First, we use the value of X where a fault-proneness model curve changes direction the most, i.e., it has maximum convexity. Then, we use the values of X where the slope has specific values: one-half of the maximum slope, and the median and mean slope in the interval between minimum and maximum slopes.

Results. We provide the theoretical underpinnings for our approach and we apply our approach to data from the PROMISE repository by building Binary Logistic and Probit regression fault-proneness models. The empirical study shows that the proposed thresholds effectively detect "early symptoms" of module faultiness, while achieving a level of accuracy in classifying faulty modules close to other usual fault-proneness thresholds.

Conclusions. Our method can be practically used for setting "early symptom" thresholds based on evidence captured by statistically significant models. Also, the thresholds depend on characteristics of the models alone, so project managers do not need to devise the thresholds themselves. The proposed thresholds correspond to increasing risk levels, so project managers can choose the threshold that best suits their needs in a risk-averse framework.

1. Introduction

The assessment of the fault-proneness of software modules can help software practitioners along the software life cycle and software researchers in the evaluation and improvement of software development techniques. By the term "module," we denote any piece of software (e.g., routine, method, class, package, subsystem, system) in the remainder of the paper.

Fault-proneness is an "external" software attribute [1] that can be defined as the probability that a software module contains a fault. Fault-proneness, according to Measurement Theory [2,3], is quantified by means of estimation models (which we call fault-proneness models) whose independent variables are measures taken from modules or from the software development process. For simplicity, in this paper we only use ratio measures [2] taken on modules, which quantify "internal" software attributes like size, structural complexity, cohesion, and coupling. Also, for illustration's sake, we deal with univariate models (i.e., with one independent variable) until Section 8, where we show how to extend our results to multivariate models (i.e., with more than one independent variable). Finally, we also assume in most of the paper that a fault-proneness model $fp(X)$ is a monotonically increasing function of its independent variable X, as is often the case in Empirical Software Engineering. We highlight the differences with the case of monotonically decreasing fault-proneness models whenever needed, notably in Section 7.

* Corresponding author .
E-mail addresses: sandro.morasca@uninsubria.it (S. Morasca), luigi.lavazza@uninsubria.it (L. Lavazza).

http://dx.doi.org/10.1016/j.infsof.2017.03.005
0950-5849/

At any rate, the sheer knowledge of the value of fault-proneness of a module may not be sufficient in practice. It would be very useful to practitioners to have indications on whether a module should be considered faulty or non-faulty, i.e., on a binary scale, and not just more or less fault-prone, i.e., on a continuous scale. These indications can be used during software development to identify which modules need to undergo stricter Validation & Verification (V&V), maybe even before they are tested. Thus, these indications can be usefully based on the values of an internal measure computed on the modules. As a consequence, we need a criterion to partition modules into those that can be estimated faulty and those that can be estimated non-faulty based on the value of an internal measure X. Usually, the values of X are partitioned by means of a threshold value x_t and modules with $X \leq x_t$ are estimated non-faulty and modules with $X > x_t$ faulty.

1.1. Properties for threshold setting

Clearly, this approach is effective only if the threshold is set in a sensible way. However, the definition of the value of x_t is somewhat arbitrary. We believe that threshold setting should satisfy the following four properties.

- *TSP1* A threshold should be set to take into account the practitioners' goals: as a consequence, different ways of setting thresholds may be used in different cases.
- *TSP2* A threshold on X should be set only if there is evidence that X is related to the likelihood of having faults. Specifically, we require that a statistically significant fault-proneness model exists where X is the independent variable.
- *TSP3* A threshold may be based on the characteristics of the specific fault-proneness model. Thus, different thresholds on X may be set with different fault-proneness models for X, built with different techniques.
- *TSP4* Since X is a ratio scale, any threshold on it should be invariant up to a proportional transformation. Thus, if we transform variable X into a new variable Y, with $X = kY$ for some given k and if x_t is a threshold for X, the corresponding threshold y_t on Y is such that $x_t = k \cdot y_t$. We show in Section 9 an example of an approach that may appear to be sensible, but does not satisfies these requirements.

Here is a possible two-step approach that can be used to satisfy these properties.

(A) First, a specified maximum acceptable threshold value of fault-proneness fp_t is set, based on a number of factors, e.g., economic ones, related to the costs associated with 1) monitoring or acting on modules that are not actually faulty and 2) not monitoring or not acting on modules that are actually faulty.

(B) Second, the value of fp_t is used to partition the values of X into two classes: the values x such that $fp(x) > fp_t$ are estimated to be related to possibly faulty modules and the others are estimated to be related to possibly non-faulty modules. As $fp(x)$ is monotonically increasing, there exists one value x_t such that $fp(x_t) = fp_t$. For instance, consider Fig. 1(a) of Section 10.1, where several candidate values for fp_t are shown. Setting $fp_t = 0.5$ (shown in the figure as the "Fifty" threshold) entails setting a threshold on *CBO* equal to $cbo_t = 10$.

This is one of the ways the information about the relationship between X and fault-proneness can be used, as recommended by Bender in epidemiological studies [4], and also referenced in some of the literature on the setting of thresholds in Empirical Software Engineering [5,6].

Bender also suggests an alternative approach, in which a threshold on X is defined to mark the point where the slope of the estimation model becomes "too steep." Thus, the focus is no longer on $fp(x)$, but on $fp'(x)$, which is the first derivative of $fp(x)$.

1.2. Goal of the paper

The goal of this paper is to show how $fp'(x)$, i.e., the slope of $fp(x)$, can be used to set thresholds on X, by introducing two different kinds of proposals. In both cases, the underlying idea is that during development, some software managers may prefer to closely monitor a module, or even take action on it, not when its fault-proneness is already past a specified fault-proneness threshold, but at the "early symptoms" of possible faultiness, based on the *variations* in a fault-proneness model, rather than the value of fault-proneness itself. Our two approaches look at those variations in two different ways.

- Suppose that a software module evolves over time and that the value of X increases. Typically, the graph of $fp(x)$ shows that, at the beginning of the evolution of the module, i.e., for small values of X, even fairly large variations in X imply small variations in fault-proneness, i.e., $fp(x)$ is rather "flat." For instance, Fig. 1(b) shows that fault-proneness stays practically nil for values of *CBO* (Coupling Between Objects) up to about 4. As it increases, however, X will reach a value past which $fp(x)$ begins to depart too fast from the low-risk "flat," "safe" area.
  Thus, our first proposal is to define the threshold as the value of X in which $fp(x)$ bends the most, i.e., the point that best separates the region of the x-axis in which $fp(x)$ is "flat" from the rest of the x-axis. This point is the one in which the convexity of $fp(x)$ is maximum.
  Note that $fp(x)$ is not necessarily "too steep" when it has its maximum convexity. For instance, as Fig. 1(b) shows, $fp(x)$ is steepest when $X = 10$, but waiting to act on a module until X reaches 10 or a value like 8—which may be "too steep" because it is "close" to 10—may be regarded as too late (in Fig. 1(b), X is 6.5 when convexity is maximum). Note also that $fp(x)$ is not necessarily "too high" when it has its maximum convexity. Actually, since we are interested in identifying the "early symptoms" of possible faultiness, we want to identify them well before $fp(x)$ becomes "too high."
- Other software managers may instead prefer to closely monitor a module, or even take action on it, when X reaches a point where even small variations of X no longer imply relatively harmless fault-proneness variations. For instance, possible small changes to a software module may entail large changes in its fault-proneness, and this may be viewed as an "early symptom" of potential problems with the module.
  Our second proposal is therefore to define thresholds as the points in which the slope of $fp(x)$ assumes specific values. One possibility is to choose the point where the slope is a specified fraction of the maximum slope of $fp(x)$, e.g., one-half. Other options consist in setting the threshold where the slope value is the median or the mean of the slopes in the interval between the two points with the minimum and the maximum slopes. In all these cases, the idea is that the threshold divides the values of X into two disjoint subsets: values characterized by "flat" slopes (less than one half of the maximum, or less than the median or mean slope) and values characterized by "steep" slopes (more than one half of the maximum, or more than the median or mean slope). Software project managers need to chose one of these criteria, based on a number of factors, including economic ones, like in step (A) above.

Thus, our slope-based proposals reflect two truly different approaches to managing risk-aversion.

Please cite this article as: S. Morasca, L. Lavazza, Risk-averse slope-based thresholds: Definition and empirical evaluation, Information and Software Technology (2017), http://dx.doi.org/10.1016/j.infsof.2017.03.005