The Journal of Systems and Software 93 (2014) 24–41

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Controversy Corner

Predicting software defects with causality tests

Cesar Couto[a,b,*], Pedro Pires[a], Marco Tulio Valente[a], Roberto S. Bigonha[a], Nicolas Anquetil[c]

[a] Department of Computer Science, UFMG, Brazil
[b] Department of Computing, CEFET-MG, Brazil
[c] RMoD Team, INRIA, Lille, France

ARTICLE INFO

Article history:
Received 6 May 2013
Received in revised form 13 January 2014
Accepted 14 January 2014
Available online 5 February 2014

Keywords:
Defect prediction
Causality
Granger test

ABSTRACT

In this paper, we propose a defect prediction approach centered on more robust evidences towards causality between source code metrics (as predictors) and the occurrence of defects. More specifically, we rely on the Granger causality test to evaluate whether past variations in source code metrics values can be used to forecast changes in time series of defects. Our approach triggers alarms when changes made to the source code of a target system have a high chance of producing defects. We evaluated our approach in several life stages of four Java-based systems. We reached an average precision greater than 50% in three out of the four systems we evaluated. Moreover, by comparing our approach with baselines that are not based on causality tests, it achieved a better precision.

1. Introduction

Defect prediction is a central challenge for software engineering research (Basili et al., 1996; Zimmermann et al., 2008; D'Ambros et al., 2010; Kamei et al., 2013). The goal is to discover reliable predictors that can indicate in advance those components of a software system that are more likely to fail. Clearly, this information is of central value for software quality assurance. For example, it allows quality managers to allocate more time and resources to test—or even to redesign and reimplement—those components predicted as defect-prone.

Due to its relevance to software quality, various defect prediction techniques have been proposed. Essentially, such techniques rely on different predictors, including source code metrics (e.g., coupling, cohesion, size) (Basili et al., 1996; Subramanyam and Krishnan, 2003; Nagappan et al., 2006), change metrics (Hassan, 2009), static analysis tools (Nagappan and Ball, 2005; Araujo et al., 2011; Couto et al., 2013), and code smells (D'Ambros et al., 2010). Specifically, in a recent paper we reported a study showing the feasibility of using causality tests to predict defects in software systems (Couto et al., 2012). We relied on a statistical hypothesis test proposed by Clive Granger to evaluate whether past changes to a given source code metrics time series can be used to forecast changes in defects time series. Granger test was originally proposed to evaluate causality between time series of economic data (e.g., to show whether changes in oil prices cause recession) (Granger, 1969, 1981). Although extensively used by econometricians, the test was also used in bioinformatics (to identify gene regulatory relationships, Mukhopadhyay and Chatterjee, 2007) and recently in software maintenance (to detect change couplings spread over an interval of time, Canfora et al., 2010). In our study, we found that 64–93% of the defects in four well-known open-source systems were detected in classes with a Granger-positive result between the respective time series of source code metrics and defects.

In this paper, we leverage this initial study by proposing and evaluating a defect prediction model based on causality tests. More specifically, we not only report that Granger-causalities are common between time series of source code metrics and defects (which is essentially a theoretical result), but we also propose a model that relies on this finding to trigger alarms as soon as changes that are likely to introduce defects in a class are made (i.e., a model that can contribute effectively to software quality assurance practices). Fig. 1 provides details on our approach for defect prediction. In a first step, we apply the Granger test to infer possible Granger-causalities between historical values of source code metrics and the number of defects in each class of the system under analysis. In this first step, we also calculate a threshold for variations in the values of source code metrics that in the past Granger-caused defects in such classes. For example, suppose that a Granger-causality is found between changes in the size of a given class in terms of lines

* Corresponding author at: Department of Computing, CEFET-MG, Brazil. Tel.: +55 3186612513.
E-mail addresses: cesarfmc@dcc.ufmg.br, cesar@decom.cefetmg.br, cesarfmc@gmail.com (C. Couto), ppires@dcc.ufmg.br (P. Pires), mtov@dcc.ufmg.br (M.T. Valente), bigonha@dcc.ufmg.br (R.S. Bigonha), nicolas.anquetil@inria.fr (N. Anquetil).

0164-1212/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2014.01.033

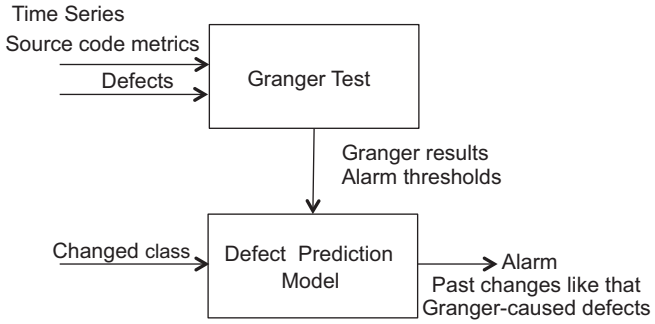

Fig. 1. Proposed approach to predict defects.

of code (LOC) and the number of defects in this class. Considering previous changes in this specific class, we can establish for example that changes adding more than 50 lines of code are likely to introduce defects in this class (more details on how such thresholds are calculated in Section 3.3). Using these thresholds and the Granger results calculated in the previous step, a defect predictor analyzes each change made to a class and triggers alarms when similar changes in the past Granger-caused defects.

Regarding our initial study, we also extended a dataset proposed to evaluate defect prediction approaches, by almost doubling the number of source code versions included in this dataset. Finally, we evaluated our approach in several life stages of four open-source systems included in the aforementioned dataset. Our approach reached an average precision greater than 50% considering three out of the four systems we evaluated. Moreover, our results show that the precision of the alarms changes with time. For example, for the Eclipse JDT Core, we achieved an average precision of 58% considering 144 models covering seven years of the system's history, and including a minimal and maximal precision of 27% and 90%, respectively. On the other hand, we were not able to predict all defects using times series of source code metrics. On average, we achieved recall rates ranging from 13% (Equinox Framework) to 31% (Lucene). In fact, we argue that it is not feasible to expect that alarms based on source code metrics variations can cover the whole spectrum of bugs reported to a system. Finally, we show that our models outperform models that trigger alarms without considering Granger-causality or that are based on linear regression techniques.

The remainder of this paper is organized as follows. We start with an overview on Granger Causality (Section 2). Next, we describe the steps to build the proposed model (Section 3), including the time series extraction, the application of the Granger test, and the identification of thresholds in metrics variations that may lead to defects. Section 4 describes our dataset including time series of source code metrics and defects for four real-world systems (Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, and Lucene). Section 5 describes a feasibility study designed to illustrate and to evaluate the application of Granger on defects prediction. We present an evaluation of the proposed model in Section 6. Section 7 discusses related work, and Section 8 concludes the paper.

2. Granger causality

In this section, we start first by describing a precondition that Granger requires the time series to follow (Section 2.1). Next, we present and discuss the test (Section 2.2).

2.1. Stationary time series

The usual pre-condition when applying forecasting techniques—including the Granger test described in the next subsection—is to require a stationary behavior from the time series (Fuller, 1994). In stationary time series, properties such as mean and variance are constant over time. Stated otherwise, a stationary behavior does not mean the values are constant, but that they fluctuate around a constant long run mean and variance. However, most time series of source code metrics and defects when expressed in their original units of measurements are not stationary. The reason is intuitively explained by Lehman's Law of software evolution, which states that software measures of complexity and size tend to grow continuously (Lehman, 1980). This behavior is also common in the original domain of Granger application, because time series of prices, inflation, gross domestic product, etc. also tend to grow along time (Granger, 1981).

When the time series are not stationary, a common workaround is to consider not the absolute values of the series, but their differences from one period to the next one. More specifically, suppose a time series $x(t)$. Its *first difference* $x'(t)$ is defined as $x'(t) = x(t) - x(t-1)$.

Example 1. To illustrate the notion of stationary behavior, we will consider a time series that represents the number of methods (NOM), extracted for the Eclipse JDT Core system, in intervals of bi-weeks, from 2001 to 2008. Fig. 2(a) illustrates this series. As we can observe, the series is not stationary, since it has a clear growth trend, with some disruptions along the way. Fig. 2(b) shows the first difference of NOM. Note that most values are delimited by a constant mean and variance. Therefore, NOM in first difference has a stationary behavior.

2.2. Granger test

Testing causality between two stationary time series x and y, according to Granger, involves using a statistical test—usually the F-test—to check whether x helps to predict y at some stage in the future (Granger, 1969). If this happens, we can conclude that x Granger-causes y. The most common implementation of the Granger Causality Test uses bivariate and univariate auto-regressive models. A bivariate auto-regressive model includes past values from the independent variable x and from the dependent variable y. On the other hand, a univariate auto-regressive model considers only past values of the variable y.

To apply Granger, we must first calculate the following bivariate auto-regressive model (Canfora et al., 2010):

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \cdots + \alpha_p y_{t-p} + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_p x_{t-p} + u_t \quad (1)$$

where p is the auto-regressive lag length (an input parameter of the test) and u_t is the residual. Essentially, p defines the number of past values—from both x and y—considered by the regressive models. Furthermore, Eq. (1) defines a bivariate model because it uses values of x and y, limited by the lag p.

To test whether x Granger-causes y, the following null hypothesis must be rejected:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

This hypothesis assumes that past values of x do not add predictive power to the regression. In other words, by testing whether the β coefficients are equal to zero, the goal is to discard the possibility that the values of x contribute to the prediction.

To reject the null hypothesis, we must first estimate the following auto-regressive univariate model (i.e., an equation similar to Eq. (1) but excluding the values of x):

$$y_t = \gamma_0 + \gamma_1 y_{t-1} + \gamma_2 y_{t-2} + \cdots + \gamma_p y_{t-p} + e_t \quad (2)$$