Chapter IX

A Practical Software Quality Classification Model Using Genetic Programming

Yi Liu, Georgia College & State University, USA
Taghi M. Khoshgoftaar, Florida Atlantic University, USA

Abstract

A software quality estimation model is an important tool for a given software quality assurance initiative. Software quality classification models can be used to indicate which program modules are fault-prone (FP) and not fault-prone (NFP). Such models assume that enough resources are available for quality improvement of all the modules predicted as FP. In conjunction with a software quality classification model, a quality-based ranking of program modules has practical benefits since priority can be given to modules that are more FP. However, such a ranking cannot be achieved by traditional classification techniques. We present a novel software quality classification model based on multi-objective optimization with genetic programming (GP). More specifically, the GP-based model provides both a classification (FP or NFP) and a quality-based ranking for the program modules. The quality factor used to rank the modules is typically the number of faults or defects associated with a module. Genetic programming is ideally suited for optimizing
multiple criteria simultaneously. In our study, three performance criteria are used to evolve a GP-based software quality model: classification performance, module ranking, and size of the GP tree. The third criterion addresses a commonly observed phenomena in GP, that is, bloating. The proposed model is investigated with case studies of software measurement data obtained from two industrial software systems.

Introduction

Software quality assurance is a vital component of any given software development process and consists of various techniques and methods used toward software quality improvement. A software quality model aims to find the underlying relationship between software measurements and software quality. The problem of predicting, during the software development process, which program modules are likely to be problematic is of practical importance in software engineering practice (Khoshgoftaar & Seliya, 2004).

Any given software development project usually has a finite amount of resources it can expend. In addition, the amount of resources allocated for software quality improvement is typically a fraction of the total budget. Therefore, it is of practical importance that the software development team seeks effective ways to achieve the best possible software quality within the allocated budget. One of the commonly-adopted methods for improving software quality is to classify program modules into two risk-based groups, such as fault-prone (FP) and not fault-prone (NFP) (Briand, Melo, & Wust, 2002; Ping, Systa, & Muller, 2002; Runeson, Ohlsson, & Wohlin, 2001). Subsequent to the calibration of a software quality classification model, all the modules predicted as FP are subjected to software quality improvement activities. A software quality model can also be built to predict the number of defects in program modules (Ganesan, Khoshgoftaar, & Allen, 2000; Khoshgoftaar & Seliya, 2002b, 2003).

In related literature, various techniques have been explored for calibrating metrics-based software quality classification models (Schneidewind, 2002). Some commonly used techniques include logistic regression (Khoshgoftaar & Allen, 1999), classification trees (Khoshgoftaar & Allen, 2001; Khoshgoftaar & Seliya, 2002a), case-based reasoning (Khoshgoftaar, Cukic, & Seliya, 2002), and computational intelligence-based methods (Pizzi, Summers, & Pedrycz, 2002; Reformat, Pedrycz, & Pizzi, 2002; Suarez & Lutsko, 1999). The definition of what constitutes a FP program module is dependent on the project management team and the software quality objectives of the system under consideration. Usually, the number of software faults or defects associated with a program module determines whether it is to be considered as high-risk (FP) or low-risk (NFP). In cases where defect data is not available, the amount of code churn (changes to source code to fix a problem) for a given module may determine whether it is FP or NFP.

The underlying assumption behind a FP-NFP software quality classification model is that all program modules predicted as FP will be subjected to inspections. Since each development organization has a pre-defined procedure for conducting software quality improvements, all the FP modules will be subjected to the same inspection process. This implies that the software project should have enough resources to inspect all the FP modules. Though
Cognitive Computational Models of Emotions and Affective Behaviors
www.igi-global.com/article/cognitive-computational-models-emotions-affective/72879?camid=4v1a