Modelling software quality with GP

Matthew Evett, Taghi Khoshgoftaar, Pei-der Chien, Edward Allen
Dept. Computer Science & Engineering
Florida Atlantic University
Boca Raton, Florida 33431
(matt,taghi,chienp,allene)@cse.fau.edu

Abstract

Software development managers use software quality prediction methods to determine to which modules expensive reliability techniques should be applied. This paper describes a genetic programming (GP) based system that classifies software modules as "faulty" or "not faulty", allowing the targetting of modules for reliability enhancement. The system is validated via a case study using software quality data from a very large industrial project. The demonstrated quality of the system is such that the system is being integrated into a commercial software quality management system.

We used GP to create models that predict the number of faults expected in sets of modules, and use these predictions to rank the modules. With a given rank-order of the modules, the software manager must decide which modules to submit to reliability testing. This is done by selecting a threshold (or cut-off), c, a percentage. In effect, the threshold delineates the set of modules that are considered "faulty" from those that are considered "not faulty". The top c percent of the modules according to the ranking (i.e., those designated "faulty") are chosen for testing. The difficulty is in choosing c so as to minimize the cost of not testing faulty modules (*Type II errors*) or needlessly testing non-faulty modules (*Type I errors*).

The basic unit for software quality modeling is an *observation*, which is a software module represented by a tuple of software measurements, $\mathbf{x}_j$, for observation j. The $\mathbf{x}_j$ values are vectors of the form $< x_{j:1}, x_{j:2}, \ldots, x_{j:n} >$, whose components are discrete or real-valued software metrics. The dependent variable of a model is the *quality factor*, $\mathbf{y}_j$, for each observation j. The *quality factor* usually is the number of faults in each software module for this study. The programs resulting from our GP system are the *models*, which estimate the $\mathbf{y}_j$.

The function set consisted of a standard collection of arithmetic functions, and the terminal set consisted of the available software product and process metric variables (the independent variables of the data sets) and the ephemeral random constant generator function.

The *fitness* of individual i is defined as the *type II* error rate of individual i:

$$f_{adj}(i) = 1 - typeII_i(p) \tag{1}$$

where p, the "training cut-off", is a cut-off percentage value, and $typeII_i(p)$ is the percentage of modules that were misclassified using that cut-off. We used an impartially selected subset of observations taken from a very large legacy telecommunications system (more than ten million lines of code, and over 3500 software modules) as the test cases for our GP system. The remaining observations were used for validation.

Each best-of-run was applied to the validation set of observations to achieve a ranking of those modules. We then calculated the Type I and Type II error rates for a set of proposed cut-offs. The manager can then look at the error rates obtained by the various cut-off values to determine which cut-off value obtains error rates that optimize the cost of reliability analysis, but which are within the financial and man-power resource constraints.

This work, in combination with our related previous work (see GP98) has demonstrated the effectiveness of GP to generate accurate software quality models in real-world, industrial domains.

This work was supported in part by a grant from Nortel. The findings and opinions in this study are the authors', and are not necessarily those of the sponsor.