Identifying Source Code Metrics to Improve Quality Predictive Models Using Genetic Algorithms

Rodrigo Vivanco
Dept. Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
(204) 983-7393

rvivanco@cs.umanitoba.ca

ABSTRACT

In order to develop high quality software in a timely and cost-effective manner, it is essential to identify and correct faulty components. Predictive models are used to classify modules as potentially faulty and in need of further inspection. Source code metrics can be used as input features for the classifier, however, there exists a large number of measures that capture different aspects of coupling, cohesion, inheritance, complexity and size. In large dimensional feature spaces some of the metrics may be irrelevant or redundant. Feature selection is the process of identifying a subset of features that improve the classifier's discriminatory performance. The focus of this study is to explore the efficacy of evolutionary algorithms to search for an optimum combination of object-oriented source code metrics to improve a predictive model's ability to identify fault-prone classes.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *object-oriented, quality predictive models, feature subset selection.*

General Terms

Measurement, Design, Reliability, Experimentation.

Keywords

Predictive Models, Object Oriented Metrics, Genetic Algorithm

1. INTRODUCTION

Improving a system's reliability and maintainability will reduce overall development costs and enhance product quality [1]. Predictive models have been a focus of research in empirical studies of software systems [2]. Many of these studies attempt to associate a causal relationship between a system's structural metrics, obtained from source code measures, with external objective measures of quality such as component faults. Being able to identify system modules that are likely to be fault prone would assist project managers take mitigating actions (such as increased source code inspection, assigning the best developers to such modules or performing more exhaustive testing) in order to improve its reliability and decrease corrective and preventive maintenance costs.

ACM SigSoft '06, November 5–11, 2006, Portland, Oregon, USA.

Many organizations have adopted object-oriented technologies for product development and numerous studies have been conducted into the formulation of metrics to quantify various aspects of a system's design and implementation. Chidamber and Kemerer [3] introduced six metrics that attempt to capture the coupling, cohesion, inheritance and complexity in object-oriented designs. Briand et al. have identified at least 15 different cohesion and 30 coupling measures in the literature [4, 5].

Many of the proposed structural metrics attempt to measure the structural aspects of the system, namely cohesion, coupling, inheritance, complexity and size. As such, most metrics exhibit various levels of correlation. For example, complexity measures can be associated with simple size measures such as the number of lines of code. Using correlated measures decreases the accuracy of multivariate predictive models when identifying fault-prone components.

Various techniques can be used to reduce a large collection of metrics to a smaller set that exhibit strong discriminatory powers in the identification of fault-prone classes [6]. One approach is to use univariate linear regression to identify metrics as potentially useful predictors of faulty modules [7]. Secondly, if the number of available metrics is small enough, stepwise selection can be used. A third approach is to use principal component analysis [8], which are linear combinations of normalized measures that capture the maximum variance of the data [9].

A new approach using evolutionary computational methods, in particular genetic algorithms [10], is proposed as a means of heuristically identifying combinations of source code metrics that improve the performance of predictive models. Genetic algorithms have been used in the medical field for feature subset selection [11]. In software engineering they have been used to modify models but not to identify source code metrics to be used in predictive models [12].

2. PROBLEM STATEMENT

Faulty components degrade product quality and increase development costs. Source code measures are used as features for predictive models of potentially defective components. However, there exists a large number of structural metrics that measure different elements of the basic principles of coupling, cohesion, inheritance, complexity and size, some of which may be irrelevant or redundant. In the absence of theoretical reasons to choose a particular set of metrics and the observed association of

the various metrics, there is a need to find an effective method of feature subset selection in order to improve the performance of predictive models and illuminate the potential theoretical underpinnings of the significant metrics. The main hypothesis to test is that classification models that use genetic algorithm driven metrics outperform classifiers that use metrics obtained from principal component analysis.

2.1 Background

The curse of dimensionality [13] is a well-known problem in machine learning. As the number of dimensions in the feature space grows, it can degrade classifier performance, in particular when the size of the training set is small, which is usually the case with empirical software engineering.

One approach is to use univariate linear regression analysis to identify potentially useful predictors to utilize in the model; however, it does not take into account the discriminatory power of using combined measures.

If the number of available metrics is not prohibitively high, stepwise selection/elimination could be used with the classifier. In forward stepwise selection, each feature is added to the model, one at a time, and if it improves its performance it is kept, otherwise it is rejected. It could be that a rejected feature would improve performance in combination with a metric not yet included in the model. But since it has been rejected, it will not be taken into account.

Backward stepwise elimination starts with all the features. Each feature is tentatively removed and the model tested. The feature that has the least positive impact on the model's performance is permanently removed from the model. Since the features are removed one at a time, this approach is sensitive to local minima, as the process stops when model performance is not improved.

Another approach is to use principal component analysis (PCA), based on linear combinations of the metrics that maintain the maximum variance of the dataset. With PCA, all the metrics are considered when calculating loading factors. The loading factor of the metrics in one dimension (a principal component) is an indication of the importance of the metric in explaining the variance. Metrics with a loading factor above a user-defined threshold are then used with the predictive model. It is assumed that the metrics that explain the maximum variance will result in more accurate models. Setting the loading factor threshold is another limitation of PCA, as it user dependent. As well, with PCA, the same metrics subset will be used for any type of predictive model. A set that works well with a non-linear classifier may not do so well with a linear classifier. It has been shown that using PCA with different datasets also results in different metrics being selected [8].

3. PROPOSED SOLUTION

A new approach using evolutionary computational methods, in particular genetic algorithms, is proposed as a means of identifying combinations of source code metrics that improve classifier performance. Genetic algorithms (GA), based on the evolutionary mechanics of selection of the fittest, are used in optimization and search problems where a clear analytical solution is not readily available. With the large number of available source metrics and the various confounding effects of related metrics, a GA can be used to heuristically search the solution space for a combination of metrics with that improve the predictive performance for different types of classifiers.

Following the guidelines of various authors [6, 14, 15] an empirical study into software metrics will be conducted. The proposed research will have multiple outcomes but the main hypothesis to be tested is: genetic algorithms provide a feasible and more robust alternative to principal component analysis for selecting subsets of object-oriented metrics in order to improve the predictive power of models in identifying fault-prone classes.

An objective measure of quality is the preferred dependent variable for predictive models. Fault proneness is a frequently used external measure of quality, in particular, any class containing one or more defects is considered fault prone. This study will use the fault-proneness of a class as the dependent variable. Each object-oriented class in the available datasets will be identified as fault-prone or not-fault-prone by mining CVS repositories and bug reports. For this study, PCA will be employed to identify a metrics subset to be used as the baseline for comparative analysis to metrics subsets obtained with the genetic algorithm.

3.1 Genetic Algorithms

A genetic algorithm (GA) heuristically searches for an optimal set of solutions to a problem by simulating evolution. In GA, a solution (a set of software metrics) is encoded in a gene and a collection of solutions (genes) makes up a population. GA uses natural selection and genetics as a basis to discover optimal solutions, i.e. genes that encode a set of software metrics that give the best classification rate. A population of solutions is modified using directed random variations and a parent selection criterion. Genetic algorithms are based on the process of Darwinian evolution; over many generations, the “fittest” individuals tend to dominate the population. In the context of this study, the fittest individual results in the best classifier performance.

The simplest way to represent a gene is to use a string of bits, where ‘0’ means the bit is off and ‘1’ means the bit is on. For this problem domain, the N metrics will be encoded in an N-bit mask vector. A zero bit means the metric is not to be used with the classifier. All the metrics with a corresponding bit set to ‘1’ constitute the metrics sub-set to evaluate with the fitness function (a predictive model). The genes of the initial population will be randomly initialized.

Various GA parameters can be adjusted that influence the effectiveness and speed of the solution search. The number of genes in a population, the mutation rate, the number of genes to keep for the next generation, and the number of generations are some of most common parameters. The number of genes in the population and the number of generations will have the largest impact on computational speed and in avoiding local minima. Larger populations are more robust, but take more time to evolve. The most computationally intensive aspect of a GA is the calculation of a gene’s fitness value (how well the predictive model performs).

Genetic algorithms lend themselves naturally to parallelization [16]. The fitness function for one gene is usually executed independently of other genes in the population. That means one process can calculate the fitness function for one gene in a population, while another process does the same for another gene in parallel. Once all the genes in the current population are

evaluated, the new population is evolved and the process is repeated for the next generation. Using a parallel GA greatly enhances the efficiency of the search and larger populations over longer generations can be used. Figure 1 illustrates the worker/manager parallel GA that has been implemented to search for optimal metrics. The manager process sends a gene to a worker process until the population is fully evaluated.

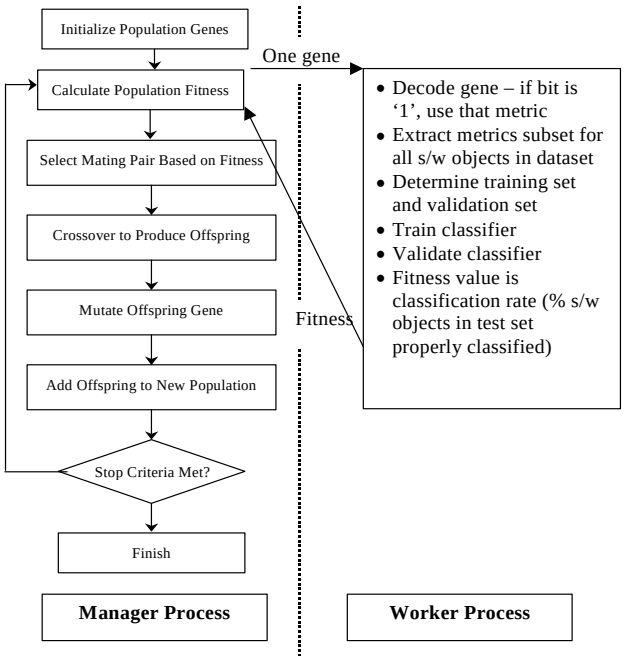

Figure 1. Basic manager/worker parallel algorithm for a GA.

3.2 Predictive Models

Multivariate linear regression is a commonly used predictive model in empirical studies. A linear discriminant analysis (LDA) classifier will be utilized in this research as the multivariate linear model. LDA is a conventional supervised classification strategy used to determine linear decision boundaries between groups while taking into account between-group and within-group variances [17]. Each software object will be labeled as fault prone or not fault-prone. The software objects in the dataset will be divided into training and test sets.

For small datasets a common approach is to train and test the classifier using leave-one-out cross-validation. In leave-one-out, the classifier is trained on all the software objects except one. The one left out is classified after the training is done. This is repeated for all the software objects in the dataset. With larger datasets v-cross or random train/test subsets could be used. With v-cross validation, the dataset is divided into v subsets. For each subset, training is done on the remaining subsets and testing is performed with the current subset.

The available datasets will be inspected and the appropriate cross validation method applied. The classification rate is the number of software in the test set correctly categorized by the LDA and will be used as a gene's fitness value to be compared with other genes in the population.

Non-linear methods have been suggested as superior since linear models are susceptible to correlations in the independent variables, something that may not be avoided in source code metrics. This study will use an artificial neural net (ANN) [17] as a non-linear classifier to compare against a linear classifier (LDA). A direct comparison between LDA and ANN can be made using the PCA derived metrics, since with PCA the same metrics subset is used with any model under consideration.

Since genetic algorithms are search driven optimizers, a different metrics subset may result for the LDA and the ANN fitness functions. A metrics subset with correlated measures that would degrade a linear classifier may not degrade a non-linear model. The purpose of using ANN is to determine if different GA metrics are identified for linear vs. non-linear models. As well, a comparison will be made of the effectiveness of GA metrics when the search is tuned for a particular classifier, as opposed to PCA, in which the same metrics set will be used with both classifiers.

Adapting the GA to use an ANN instead of LDA only involves changing the fitness function while everything else remains the same. Various parameters affect the performance of a neural net. The most influential are the number of hidden layers, the number of nodes in each layer and the type of node connections. A capable neural net architecture will be determined from the literature and used for all analysis.

3.3 Open Source Dataset

Product measures and fault data will be obtained from three related C++ open-source projects, all implemented using a similar infrastructure and coding standards [18]. ITK is an image processing toolkit for segmentation and registration. It provides an N-dimensional templated data model and image processing algorithms. VTK is a visualization toolkit and in conjunction with ITK is used to build image-processing applications. IGSTK is used with ITK and VTK to implement robust applications for image-guided surgery.

The source code metrics will be extracted using an open-source parser, if available, or a trial version of a commercial product. Descriptive statistics on the open-source datasets will be calculated to determine distribution of the measures and which software objects should be excluded from analysis, if any. The values of the source code metrics will be normalized before being used with the predictive models.

3.4 Evaluation of Results

The main hypothesis to test is that classification models that use GA driven metrics outperform classifiers that use metrics obtained from PCA. Using a classifier's classification rate is not enough to compare different models. Since the faulty software objects in the dataset are known apriori the number of false positives (Type I error) and false negatives (Type II error) can be calculated. For a two-class problem the sensitivity and specificity can be derived. The various metrics/classifiers (PCA with LDA/ANN, GA with LDA/ANN) can then be compared and the best performing combination of measures and model identified. As well, since the total number of faults is known, the completeness of the various models can be assessed and compared. The classifier with the highest classification rate that is able to detect the largest number of faults would be a superior model.

The metric subsets of the GA driven optimal solutions for LDA and ANN will be compared with each other and to the PCA metrics in order to determine if they capture the same structural

properties of the system. That is, some key metrics may appear in all subsets, and the ones that appear in only one model may in fact be different ways of measuring the same structural attribute. In the latter case, the metrics that are easier to derive from the source code should be used to build predictive models.

4. INITIAL RESULTS

A preliminary study was done with an in-house Java application for biomedical data analysis. Experts were asked to rank each class as low, medium or high in terms of difficulty to understand a class (cognitive complexity) for the purpose of maintenance. A GA was used to automatically obtain a subset of metrics that improved the classifier's (LDA) performance when compared to using all available source code metrics. When all 64 metrics where used, the classifier performance was rated at 72%, while the GA optimized classifier rated at 78.4% using a subset of 28 metrics. Future work will be done with open source systems and objective measures, such as faults found in modules, will be used.

5. CONCLUSION AND FUTURE WORK

The main contribution of this research will be a new approach for the verification of object-oriented source code metrics with respect to predictive models. A parallel genetic algorithm will be developed to perform feature subset selection of object-oriented metrics and made available to the research community. A comparison of the efficacy, and metrics used by linear and non-linear classifiers will be made. A frequent observation of empirical studies in software engineering is the lack of reproducibility. In order to alleviate this problem object-oriented open source systems will be analyzed and the full datasets made freely available to the research community.

This research is currently in the initial stages. The parallel GA has been developed and tested with the proprietary Java source code. The LDA and NN classifiers have been developed. The next step is the data mining of the open source C++ datasets to identify faulty modules. Once that is done, the source code will be parsed for metrics and preprocessed for descriptive statistics. The normalized metrics will be used with PCA to obtain the baseline feature subset for each dataset. The final step will be to run the GA with the new datasets and compare them to the PCA based models.

The author sees this research as a step towards developing tools to assist developers evaluate the quality of software products for a given set of objectives, such as fault-proneness, cognitive complexity or any external measure an organization deems important. By using search-based methods such as genetic algorithms, classifiers can be developed that are tuned to a particular organization's practices. These tools can be used to identify metrics and generate quality models using training sets and objectives acceptable to a particular organization.

6. ACKNOWLEDGEMENTS

I would like to thank Dr. Vojislav Misic for his advice on the preparation of this manuscript.

7. REFERENCES

[1] Pfleeger, S.L. and Atlee J.M. *Software Engineering, Theory and Practice, 3rd Edition*. Prentice Hall, 2006.

[2] Briand L., Arisholm E., Counsell S., Houdek F. and Thevenod-Fosse P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. *Empirical Software Engineering*, 4, 4(1999), 387-404.

[3] Chidamber S.R. and Kemerer, C.F. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20, 6 (1994), 476-493

[4] Briand L.C., Daly J.W. and Wust J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Emperical Software Engineering*, 3 (1998), 65-117.

[5] Briand L.C., Daly J.W., Wust J. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25, 1 (199), 91-121.

[6] Briand L.C. and Wuest J. Empirical Studies of Quality Models in Object-Oriented Systems. *Advances in Computers*, 56 (2002), 97-166.

[7] Basili V.R., Briand L.C. and Melo W.L. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, 22 (1996), 751-761.

[8] Nagappan N., Ball T. and Zeller A. Mining Metrics to Predict Component Failures. In *Proceedings of the International Conference on Software Engineering (ICSE '06)* (Shanghai, China, May 20-28). ACM Press, New York, NY, 2006, 452-461.

[9] Dunteman D. *Principal Component Analysis*, Sage University Press, 1989.

[10] Goldberg D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[11] Raymer M.L., Punch W.F., Goodman E.D., Kuhn L.A. and Jain A.K. Dimensionality Reduction Using Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 4, 2 (2000), 164-171.

[12] Azar D., Precup D., Bouktif S., Kegl B. and Sahraoui H. Combining and Adapting Software Quality Predictive Models by Genetic Algorithms. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02)*, 2002.

[13] Bellman R. *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[14] Perry D., Porte A. and Votta L. Empirical Studies of Software Engineering: A Roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, 2000, 345-355.

[15] Kitchenham B.A., Pfleeger S.L., Hoaglin D.C., El-Emam K. and Rosenberg J. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28, 8 (2002), 721-734.

[16] Cantu-Paz E. *Effective and Accurate Parallel Genetic Algorithms*. Kluger Academic Publishers, 2000.

[17] Duda C.D., Hart P.E. and Stork D.G. *Pattern Classification 2nd Edition*. John Wiley & Sons, 2000.

[18] http://www.itk.org