A Study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm

Parvinder S. Sandhu, Sunil Khullar, Satpreet Singh, Simranjit K. Bains, Manpreet Kaur, Gurvinder Singh

Abstract—Fault-proneness of a software module is the probability that the module contains faults. To predict fault-proneness of modules different techniques have been proposed which includes statistical methods, machine learning techniques, neural network techniques and clustering techniques. The aim of proposed study is to explore whether metrics available in the early lifecycle (i.e. requirement metrics), metrics available in the late lifecycle (i.e. code metrics) and metrics available in the early lifecycle (i.e. requirement metrics) combined with metrics available in the late lifecycle (i.e. code metrics) can be used to identify fault prone modules using Genetic Algorithm technique. This approach has been tested with real time defect C Programming language datasets of NASA software projects. The results show that the fusion of requirement and code metric is the best prediction model for detecting the faults as compared with commonly used code based model.

Keywords—Genetic Algorithm, Fault Proneness, Software Fault and Software Quality.

I. INTRODUCTION

High assurance and complex mission-critical software systems are heavily dependent on reliability of their underlying software applications. A software fault prediction is a proven technique in achieving high software reliability. Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control. Quality of software is increasingly important and testing related issues are becoming crucial for software. Although there is diversity in the definition of software quality, it is widely accepted that a project with many defects lacks quality. Methodologies and techniques for predicting the testing effort, monitoring process costs, and measuring results can help in increasing efficiency of software testing. Being able to measure the fault-proneness of software can be a key step towards steering the software testing and improving the effectiveness of the whole process.

A software fault is a defect that causes software failure in an executable product. For each execution of the software program where the output is incorrect, we observe a failure. Software engineers distinguish software faults from software failures. Faults in software systems continue to be a major problem. Many systems are delivered to users with excessive faults. This is despite a huge amount of development effort going into fault reduction in terms of quality control and testing. It has long been recognized that seeking out fault-prone parts of the system and targeting those parts for increased quality control and testing is an effective approach to fault reduction. A limited amount of valuable work in this area has been carried out previously. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. The modeling techniques applied in this paper are classical approaches: principal component analysis, discriminate analysis, logistic regression, logical Quality will be improved as more faults will be detected. Predicting faults early in the software life cycle can be used to improve software process control and achieve high software reliability. Timely predictions of faults in software modules can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults. Prediction models based on software metrics, can estimate number of faults in software modules. Software metrics are attributes of the software system and may include process, product, and execution metrics. Various attributes, which determine the quality of the software, include maintainability, defect density, fault proneness, normalized rework, understandability, reusability etc.

With real-time systems becoming more complex and unpredictable, partly due to increasingly sophisticated requirements, traditional software development techniques might face difficulties in satisfying these requirements. Future real-time software systems may need to dynamically adapt themselves based on the run-time mission-specific requirements and operating conditions. This involves dynamic code synthesis that generates modules to provide the functionality required to perform the desired operations in real-time. However, this necessitates the need to develop a real-time assessment technique that classifies these dynamically generated systems as being faulty/fault-free. A variety of software fault predictions techniques have been...
proposed, but none has proven to be consistently accurate. These techniques include statistical methods, machine learning methods, parametric models and mixed algorithms as discussed in [1]-[22].

Genetic algorithm is being successfully applied for solving both classification and regression problems. It is therefore important to investigate the capabilities of this algorithm in predicting software quality. In order to perform the analysis we validate the performance of the GA method for dataset derived from the NASA’s MDP (Metric Data Program) data repository. Hence, the aim of proposed study is to explore whether metrics available in the early lifecycle (i.e. requirement metrics), metrics available in the late lifecycle (i.e. code metrics) and metrics available in the early lifecycle (i.e. requirement metrics) combined with metrics available in the late lifecycle (i.e. code metrics) can be used to identify fault prone modules using Genetic Algorithm technique.

II. METHODOLOGY

The methodology consists of the following steps:

A. Find the structural code and requirement attributes

The first step is to find the structural code and requirement attributes of software systems i.e. software metrics. The real-time defect data sets are taken from the NASA’s MDP data repository, available online at http://mdp.ivv.nasa.gov.in named as PC1 dataset which is collected from a flight software from an earth orbiting satellite coded in C programming language, containing 1107 modules and only 109 have their requirements specified [23]. PC1 has 320 requirements available and all of them are associated with program modules. All these data sets varied in the percentage of defect modules, with the PC1 dataset containing the least number of defect modules.

B. Select the suitable metric values as representation of statement

The suitable metrics like product requirement metrics and product module metrics out of these data sets are considered. The product requirement metrics are as follows:

- Module
- Action
- Conditional
- Continuance
- Imperative
- Option
- Risk_Level
- Source
- Weak_Phrase

The product module metrics are as follows:

- Module
- Loc_Blank
- Branch_Count
- Call_Pairs
- LOC_Code_and_Comment

- LOC_Comments
- Condition_Count
- Cyclomatic_complexity
- Cyclomatic_Density
- Decision_Count
- Edge_Count
- Essential_Complexity
- Essential_Density
- LOC_Executeable
- Parameter_Count
- Global_Data_Complexity
- Global_Data_Density
- Halstead_Content
- Halstead_Difficulty
- Halstead_Effort
- Halstead_Error_EST
- Halstead_Length
- Halstead_Prog_Time
- Halstead_Volume
- Normalized_Cyclomatic_Complexity
- Num_Operands
- Num_Operators
- Num_Unique_Operands
- Num_Unique_Operators
- Number_Of_Lines
- Pathological_Complexity
- LOC_Total

C. Analyze, refine metrics and normalize the metric values

In the next step the metrics are analyzed, refined and normalized and then used for modeling of fault prediction in software systems. PC1 static code based dataset contains 43 static code metrics of which various metrics that do not impact binary classification are removed like module identifier, call pairs, condition count, cyclomatic density, Decision count, Decision density, Edge count, Essential density, etc and the remaining 22 have been used for training and testing data. In the requirement based dataset the number of Input Metrics are eight.

D. Combine Requirement and code metrics

Combining requirements and code metrics have done using inner join database operation. An inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.
Genetic algorithms are used in search and optimization, such as finding the maximum of a function over some domain space.

- In contrast to deterministic methods like hill climbing or brute force complete enumeration, genetic algorithms use randomization.
- Points in the domain space of the search, usually real numbers over some range, are encoded as bit strings, called chromosomes.
- Each bit position in the string is called a gene.
- Chromosomes may also be composed over some other alphabet than \{0,1\}, such as integers or real numbers, particularly if the search domain is multidimensional.
- GAs are called "blind" because they have no knowledge of the problem.

An initial population of random bit strings is generated.

- The members of this initial population are each evaluated for their fitness or goodness in solving the problem.
- If the problem is to maximize a function \(f(x)\) over some range \([a,b]\) of real numbers and if \(f(x)\) is nonnegative over the range, then \(f(x)\) can be used as the fitness of the bit string encoding the value \(x\).

From the initial population of chromosomes, a new population is generated using three genetic operators: reproduction, crossover, and mutation.

- These are modeled on their biological counterparts.
- With probabilities proportional to their fitness, members of the population are selected for the new population.
- Pairs of chromosomes in the new population are chosen at random to exchange genetic material, their bits, in a mating operation called crossover. This produces two new chromosomes that replace the parents.
- Randomly chosen bits in the offspring are flipped, called mutation.

The new population generated with these operators replaces the old population.

- The algorithm has performed one generation and then repeats for some specified number of additional generations.
- The population evolves, containing more and more highly fit chromosomes.
- When the convergence criterion is reached, such as no significant further increase in the average fitness of the population, the best chromosome produced is decoded into the search space point it represents.

Genetic algorithms work in many situations because of some hand waving called The Schema Theorem.

- Short, low-order, above-average fitness schemata receive exponentially increasing trials in subsequent generations."

In short genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to
optimization and search problems. Genetic algorithms are
categorized as global search heuristics. Genetic algorithms are
a particular class of evolutionary algorithms (EA) that use
techniques inspired by evolutionary biology such as
inheritance, mutation, selection, and crossover. This
Technique used the feature of random search. Random search
feature selection searches the best possible solution over a
range of data. Random features and input given produce good
result. In the beginning start with a large “population” of
randomly generated “attempted solutions” to a problem then
repeatedly do the following:

- Evaluate each of the attempted solutions
- Keep a subset of these solutions (the “best” ones)
- Use these solutions to generate a new population
- Quit when you have a satisfactory solution (or you
run out of time).

The flowchart of the Genetic algorithm used is shown
below:

![Flowchart of Genetic Algorithm Used](image)

G. Implementing the model and finding the result

The proposed approach is implemented in Visual Basic
environment and MATLAB is used to calculate the
performance of the algorithm.

H. Performance Criteria

The comparisons are made on the basis of the least value of
Accuracy, MAE and RMSE values. The mean absolute error is
chosen as the standard error. The technique having lower
value of mean absolute error is chosen as the best fault
prediction technique.

- **Mean absolute error**: Mean absolute error, MAE is the
  average of the difference between predicted and actual value
  in all test cases; it is the average prediction error [26]. The
  formula for calculating MAE is given in equation shown below:

\[
\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|
\]

Assuming that the actual output is \( a \), expected output is \( c \).

- **Root mean-squared error**: RMSE is frequently used
  measure of differences between values predicted by a model or
  estimator and the values actually observed from the thing
  being modeled or estimated [26]. It is just the square root of
  the mean square error as shown in equation given below:

\[
\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - \hat{a}_i)^2}
\]

The mean-squared error is one of the most commonly used
measures of success for numeric prediction. This value is
computed by taking the average of the squared differences
between each computed value and its corresponding correct
value. The root mean-squared error is simply the square root
of the mean-squared-error. The root mean-squared error gives
the error value the same dimensionality as the actual and
predicted values.

The mean absolute error and root mean squared error is
calculated for each machine learning algorithm.

III. RESULTS AND DISCUSSIONS

The NASA MDP datasets are used in this approach to
estimate the quality of a software product. The dataset used is
PC1. We used requirement metrics, code metrics and join the
requirement and code metrics as specified in [24] for
modeling. First, the CFS algorithm is applied to the evaluate
the worth of a subset of attributes by considering the
individual predictive ability of each feature along with the
degree of redundancy between them. The direction of the
BestFirst search is set to the Forward direction and the
maximum size of the lookup cache of evaluated subsets is set to
1. This is expressed as a multiplier of the number of
attributes in the data set. The snapshot of results of CFS
algorithm applied on Combined dataset and code based
dataset is shown in figure 3 and figure 4.

![Snapshot of Result of CFS Algorithm applied on Combined Dataset](image)
--- Attribute Selection on all input data ---

Search Method:
- Best first.
- Start set: no attributes
- Search direction: forward
- Stale search after 5 node expansions
- Total number of subsets evaluated: 153
- Merit of best subset found: 0.318

Attribute Subset Evaluator (supervised, Class (numeric): 22 Faulty/Non-Faulty):
- CFS Subset Evaluator
- Including locally predictive attributes

Selected attributes: 14, 15, 16, 18 : 4
- Metric14
- Metric15
- Metric16
- Metric18

Fig. 4 Snapshot of Result of CFS Algorithm applied on Code Based Dataset.

The CFS algorithm has proposed 6 significant attributes and 4 significant attributed in the combined dataset and code based dataset respectively for the predication of fault prone modules.

Thereafter, the significant attributes are taken further analysis. In case of the combined dataset the statistics of the attributes selected is shown below:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Min Length</th>
<th>Min Range</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric4</td>
<td>0</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>Metric12</td>
<td>0.01</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>Metric22</td>
<td>0</td>
<td>79</td>
<td>3</td>
</tr>
<tr>
<td>Metric24</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Metric29</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Metric30</td>
<td>1</td>
<td>23</td>
<td>5</td>
</tr>
</tbody>
</table>

Fig. 5 Statistics of the selected attributes for the Combined Dataset.

In case of the code based dataset the statistics of the attributes selected is shown in figure 6.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Min Range</th>
<th>Min Range</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric14</td>
<td>0</td>
<td>159</td>
<td>8</td>
</tr>
<tr>
<td>Metric15</td>
<td>0</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>Metric16</td>
<td>0</td>
<td>225</td>
<td>8</td>
</tr>
<tr>
<td>Metric18</td>
<td>1</td>
<td>538</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 6 Statistics of the selected attributes for the Code Based Dataset.

The Genetic algorithm based fault proneness prediction system is developed in Visual Basic 6.0 and the Graphical User Interface developed and loaded with the selected combined dataset and code based dataset is shown in figure 7 and 8 respectively.

Table I

<table>
<thead>
<tr>
<th>Performance Criteria</th>
<th>PCI Dataset</th>
<th>Code Based Dataset</th>
<th>Combined Code and Requirement Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>92.8765</td>
<td>97.0650</td>
<td></td>
</tr>
<tr>
<td>MAE</td>
<td>0.0712</td>
<td>0.0294</td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.2669</td>
<td>0.1713</td>
<td></td>
</tr>
</tbody>
</table>

IV. CONCLUSION

In this study we predict that knowing the fault prone data at early stages of lifecycle combined with data available during code can help the project managers to build the projects with more accuracy and it will reduce the testing efforts as faulty areas are already predicted, so these modules can be handled properly.
The results of the fusion or combined model are better with 97.0650, 0.0294 and 0.1713 as Accuracy, MAE and RMSE values respectively as compared to 92.8765, 0.0712 and 0.02669 values in case of code based model.

Hence, Data available in the early stages can help the analyst to plan the required resources as and when required for development, testing. However, further investigation can be done and the impact of attributes on the fault prediction can be found. Also, more algorithms can be evaluated and then we can find the best algorithm.

REFERENCES


