Dependency Graph and Metrics for Defects Prediction

Hesham Abandah
JUST University
Jordan-Irbid
heshama@just.edu.jo

Izzat Alsmadi
Yarmouk University
Jordan-Irbid
ialsmdi@yu.edu.jo

ABSTRACT
Software defects prediction was introduced to support development and maintenance activities and improve the software quality. Reliable defect predictors can significantly optimize the utilization of software projects’ resources and increase customers’ confidence in the developed software products.

In this paper, three different classifiers (LMT, SMO and J48) are used to study the relations between dependency collected metrics and bugs collected for the software under study. ANT open source software is used in this case study. The selection of this open source was related to the availability of source code and bug reports. Results varied between the three classifiers and showed J48 to be the best classifier in terms of predicating such correlation between dependency metrics and defects. In general the three classifiers showed that there is a high significant correlation between proposed and evaluated dependency metrics and software defects which showed that they can be used as important early predictors for the software quality in general.

Author Keywords
Software quality, software testing, coupling metrics, dependency metrics, call-graph, defects prediction.

ACM Classification Keywords
D.2.8 Software Engineering: Metrics: Product metrics.

General Terms
Software Engineering; Metrics; Product metrics.

1. INTRODUCTION
For software project management, the ability to predict software quality is important for current and future activities. It is important to plan evaluation and testing for requirements, design, implementation and testing in the lifetime of the project. Further, it can help in planning future software maintenance and evolution activities after the release of the software.

Software metrics are used to collect information related to one or more software attributes with the goal of evaluating such software. Software metrics can be related to the software product, development process, management activities, etc. For software products in particular, several aspects related to the software can be assessed. Tools can be used to collect such information automatically with the least effort and time from the users’ side. Those metrics are usually collected from the software design or code. There are important aspects for quality assurance and auditing.

Call graph and dependency metrics give indication of the connectivity and the complexity of the software components interactions. While such interactions are necessary and without such interactions, software will not be able to perform its intended tasks; nevertheless, the large number of connections between software components can be a main factor for causing its complexity. Such complexity can be eventually a major contributor to the difficulty to maintain, extend or reuse such software.

The rest of the paper is organized as the following: The next section involves a literature review for some related papers. Section three describes the methodology to develop call graph based metric tool and then based on ANT 1.7 open source code and bugs, study relation with bugs. Section four includes analysis and evaluation, section five includes experiments and the paper is concluded in section six.

2. LITERATURE REVIEW
In this literature survey section, we will list several examples of research papers in two sections. In the first one, we will list examples of papers discussed the proposal and evaluation of dependency or call graph metrics in software products evaluation. The second section is in relation with using one or more software products aspects for defects prediction.

Several research papers tried to develop metrics based on call graphs and derive dependency metrics, especially code metrics and size metrics. Values of such metrics are usually assessed based on defects prediction.

Many researchers studied software modeling and found that modeling techniques were grouped into mainly two categories: Graphical modeling techniques that use a diagram with named symbols to represent the components and arcs that connect the symbols and represent the relationships and other notations to represent the constrains. Textual modeling techniques used a standardized notations and keywords to represent software components.

The researchers in [2] considered the process of extracting call dependencies as one of the most important steps in software reengineering. They developed a tool for call graph extraction based on OINK framework. The researchers in [6] made an enhancement for the hierarchical edge bundling (HEB) technique to be used as a candidate visualization technique in their framework. An experiment is conducted to compare their enhancement (HEB) and Tulip graph visualization framework. Several large systems (Bison, Mozilla Firefox, and OINK) are analyzed to conclude with the differences between the two visualization
Software developers aim to evaluate the software in terms of quality before delivering it to customers to predict possible future bugs or defects especially for critical systems that should be as free of such bugs and defects as possible. In [7, 8] researchers used CGBR metric to evaluate the ability of software metrics to predict faults. They used a case study of software from a local company and their results showed that false alarms in prediction of defects using this metric can be reduced. In one more paper [9] their results based on large software for a telecommunication company that 70% of the defects could be detected by inspecting only 3% of the code.

3. METHODOLOGY
We developed a tool to build dynamically dependency graphs based on calls between the different code classes. [1]

4. ANALYSIS AND EVALUATION
After refining the generated CSV file that represents the data set of our research with bug attribute then it will be ready to analyze and evaluate using tool WEKA 3.7.5 as data miner tool. At this study we apply the following classifier algorithms such as J48 algorithm, Logistic Model Trees (LMT) Algorithm and Support Vector Machine Algorithm (SMO) classifier. The decision tree algorithm was chosen since we want to look at classifiers that were easy to understand, so we could see how valid the correlation between call graphs based metrics and bug.

4.1 Evaluation Measures
Table 1 shows the source code metrics that are collected for the ANT project.

<table>
<thead>
<tr>
<th>Metric Type</th>
<th>Metric Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td># of executable and non-commented line code for each function</td>
</tr>
<tr>
<td>Fan In</td>
<td># of callee function list</td>
</tr>
<tr>
<td>Fan Out</td>
<td># of called function list</td>
</tr>
<tr>
<td>CGBR</td>
<td>$(1 - d) + d \sum_i CGBR(T_i)$</td>
</tr>
<tr>
<td>IFC</td>
<td>IFC(M) = LOC(M) + [fan-in(M) x fan-out(M)]²</td>
</tr>
</tbody>
</table>

Table 1. Call graph based matrices measurements [1]

The five metrics we use at this research are related to size of the software or related to coupling and dependency between the components and functions of the application under investigation. LOC metric value represents the number of executable and non-commented lines of code. FanIn metric value for such function represents the number of function calling a given function. FanOut metric value for such function represents the number of function being called by a given function. CGBR metric is an abbreviation...
to call graph based ranking and proposed by [7, 8]. This metric depends on the page ranking algorithm that used by almost of the search engines, where the ranking methodology is adopted to functions of the software. This metric hypothesis that more frequently used functions and less used modules should have different defects and bugs characteristic. IFC metric is abbreviated to information flow complexity (IFC) and represents the measurement of the total level of information flow of given function.

4.2 Principle Component Analysis using SPSS

The purpose of this analysis is to show how metrics in developed tool correlate to each other. Table (2) described PCA analysis for call graph based metrics in developed tool results in 2 orthogonal dimension components were identified from 5 call graph based metrics that have Eigen value more than 1. According to this, medium redundancy presented among these measures. In the Table 4.2, Eigen values, the variance of the data set explained by the PC (in percent), and the cumulative variance are provided for each PC. Values above 0.6 are set in boldface. The 2 PCs capture 89.963% of the variance in the data set.

<table>
<thead>
<tr>
<th>Component</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigen value</td>
<td>3.475</td>
<td>1.023</td>
</tr>
<tr>
<td>% of Variance</td>
<td>69.498</td>
<td>20.465</td>
</tr>
<tr>
<td>Cumulative %</td>
<td>69.498</td>
<td>89.963</td>
</tr>
<tr>
<td>CGBR</td>
<td>0.961</td>
<td>-0.115</td>
</tr>
<tr>
<td>LOC</td>
<td>0.930</td>
<td>-0.115</td>
</tr>
<tr>
<td>IFC</td>
<td>-0.025</td>
<td>0.966</td>
</tr>
<tr>
<td>FanIn</td>
<td>0.868</td>
<td>0.112</td>
</tr>
<tr>
<td>FanOut</td>
<td>0.961</td>
<td>0.011</td>
</tr>
</tbody>
</table>

Table 2. Rotated Component Matrix for developed tool

The PCs are interpreted as follows:

- PC1: CGBR, LOC, FanIn, and FanOut are coupling and size metrics. We have size and coupling metrics in this dimension. This shows that there are classes with high internal methods (methods defined in the class) and external methods (methods called by the class). This means coupling is related to number of methods and attributes in the class.
- PC2: IFC measure the total level of information flow of a module and reflect the degree of flow complexity among classes.

5. EXPERMENTS

At the first step, we collect the source code for the application of the study, ANT 1.7. We enter the source code for the application to a developed C# tool in order to generate call graph model for it. After that, the developed tool computes the desired metrics for each function extracted. Then compute the same metrics to classes and output the results into CSV file that represent the data set to be tested. The next step is refining the data set with bug report related to each application under investigation. Finally, evaluate the value of the metrics in terms bug and defect detection.

The format of the data set should be ARRF file, since the classifier algorithms such as J48 algorithm and MSP algorithm accepts only the files with that format. The accuracy is calculated with ten-fold cross validation. The attributes of the file listed in the figure (1)

- @attribute 'Number' 'numeric'
- @attribute 'CGBR' 'numeric'
- @attribute 'Line_of_code' 'numeric'
- @attribute 'IFC' 'numeric'
- @attribute 'Fan_In' 'numeric'
- @attribute 'Fan_Out' 'numeric'

Figure 1. Attributes of the data set

The attribute bug is classified into three categories based on the number of bugs for each class. Table (3) illustrate the all types of bugs.

<table>
<thead>
<tr>
<th>Bug Categories</th>
<th>Metric Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>VL == 0 errors / L == 1 error / M == 2 errors / H == 3 errors / VH == &gt; 3 errors</td>
</tr>
<tr>
<td>Two</td>
<td>L == 0 errors / M == 1 - 2 errors / H == &gt; 2 errors</td>
</tr>
<tr>
<td>Three</td>
<td>False == no errors / True == exist errors</td>
</tr>
</tbody>
</table>

Table 3. Categories of bugs

5.1 Experimental Results

The results of experiments show that there is an obvious correlation between the call graph based metrics and the bug and defects of the application. Table (4) will summarize all the result of the experiment.

<table>
<thead>
<tr>
<th>Bug Category</th>
<th>Category One</th>
<th>Category Two</th>
<th>Category Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application name</td>
<td>Ant 1.7</td>
<td>74.513%</td>
<td>73.539%</td>
</tr>
</tbody>
</table>

Table 4. Summary of the experiments results in terms of bug categories
As we show in the figure (2) that correlation between bug and the desired software metrics will be high when we split the bug class into small number of categories, like category three that split the bug class into two categories. So we take category three as criteria to compare the J48 classifier on the applications under investigation output to other classifier output such as Logistic Model Trees (LMT) classifier and Support Vector Machine Algorithm (SMO) classifier.

Finally, we make some normalization to our data set by excluding the non public functions such as private and protected functions from the computation of the call graph metrics for the applications under investigation, and we list the results of analysis at Table (6)

As we show in the Table (5) and figure (3) the results of three classifier algorithm are approximately have similar values, where that leads us to conclude that correlation is very high between the call graph metrics and bugs of the application under investigation.

<table>
<thead>
<tr>
<th>Classifier algorithm</th>
<th>J48</th>
<th>LMT</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant 1.7</td>
<td>75.6494%</td>
<td>77.4351%</td>
<td>75.8117%</td>
</tr>
</tbody>
</table>

Table 5. Summary of the experimental results of data set excluding non-public functions

As we show in the Table (6) and figure (4) the results of three classifier algorithm are approximately have similar values, where that leads us to conclude that correlation is very high between the call graph metrics that computed without non public functions and bugs of the application under investigation.

<table>
<thead>
<tr>
<th>Classifier algorithm</th>
<th>J48</th>
<th>LMT</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant 1.7</td>
<td>78.9861%</td>
<td>76.7857%</td>
<td>75.6494%</td>
</tr>
</tbody>
</table>

Table 6. Summary of the experimental results of data set excluding non-public functions

After studying the results in Table (5) and results in Table (6), we show that the percentage of the proposed correlation between call graph based metrics and bugs in software design is raised up when the J48 classifier algorithm is used.

6. CONCLUSION

Coupling metrics can be significant indicators on the complexity of the software products. They indicate the level of interaction between software components. In software maintenance and software quality, techniques are used to predict software quality using current software products, tools or samples of those products. The goal is to be able to predict quality of the developed software with the least amount of possible human effort.
In this paper, we used the information collected from several code coupling or dependency metrics to evaluate their correlation with software defects and hence their ability to predict the current or future existence of those defects. ANT open source code is used for the evaluation study. Both metrics and bug reports are collected automatically based on a locally developed tool. Results showed a significant correlation between evaluated metrics and software bugs. Given that the whole process is automated, those techniques can provide important assets for quality assurance and auditing.

References