An Investigation of the Relationships between Lines of Code and Defects

Hongyu Zhang

Tsinghua University

Beijing, China

2009. 9
Software Metrics

- Science and engineering can be neither effective nor practical without measurement.
- We measure software in order to better understand its status and to control its behavior.

“You cannot control what you cannot measure”
Metrics-based quality analysis

- It is widely believed that there are relationships between external software characteristics (e.g., quality) and internal product attributes.
- Discovering such relationships has become one of the objectives of software metrics.
- Software metrics and software quality:
  - Metrics can be used for quality control
  - Metrics can be used for defect prediction
LOC-based quality analysis

- There are many code attributes:
  - Complexity metrics (Vg, Number of functions, etc)
  - AST metrics (number of if statements, blocks, etc)
  - ...

- Many defect prediction models are built on top of these metrics.

- We investigate the relationship between LOC and defects, and perform defect prediction based on LOC.
  - LOC: Lines of Code, the simplest code metric
  - LOC has strong correlations with other code metrics
Dataset

- Eclipse dataset
  - Contain measurement and defect data for Eclipse versions 2.0, 2.1 and 3.0.
- Defect data
  - mined from Eclipse’s bug databases and version achieves.
  - pre-release defects (defects reported in the last six months before release)
  - post-release defects (defects reported in the first six months after release).
- Measurement data
  - contain 198 code metrics, including complexity metrics and AST metrics
Empirical Analysis of LOC

- An empirical analysis of the program LOC and defects (Zhang, APSEC’07)
  - We studied sizes of 18 large open source Java systems.
  - A small number of programs are very large, but a large number of programs are small.
  - We find that the distribution of LOC can be formally represented using the lognormal functions.
  - We call this phenomenon the small program phenomenon.
The Distribution of LOC in Eclipse/1

- For Eclipse, we rank the programs by their size (from the largest to smallest), and observe the same *small program phenomenon*.

- Most of programs are small. For Eclipse 3.0:
  - 38.03% of the programs are smaller than 32 LOC
  - 56.42% of the programs are smaller than 64 LOC

- Still, a small number of very large programs:
  - 4.39% of programs are larger than 512 LOC
  - 1.13% of programs are larger than 1024 LOC.
The Distribution of LOC in Eclipse/2

- The lognormal distribution of LOC:

\[ f(x) = \frac{1}{\sigma x \sqrt{2\pi}} \exp\left( - \frac{(\ln x - \mu)^2}{2\sigma^2} \right), \]

- The lognormal distribution of program sizes reveals the regularity behind software construction.

- The skewed distribution of program size also implies that the distribution of defects across programs is skewed.

<table>
<thead>
<tr>
<th>Eclipse</th>
<th>( \mu )</th>
<th>( \sigma )</th>
<th>( R^2 )</th>
<th>( S_e )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>3.9006</td>
<td>1.3451</td>
<td>0.9979</td>
<td>0.0161</td>
</tr>
<tr>
<td>2.1</td>
<td>3.9383</td>
<td>1.3621</td>
<td>0.9978</td>
<td>0.0166</td>
</tr>
<tr>
<td>3.0</td>
<td>3.9006</td>
<td>1.3744</td>
<td>0.9976</td>
<td>0.0171</td>
</tr>
</tbody>
</table>
Correlation between LOC and defects

- We measure the Spearman correlation between LOC and Number of Defects
- To statistically test if there is relationship between LOC and defects, we make the following hypotheses:

  \[ H_0: \text{there is no relationship between LOC and Number of Defects.} \]
  \[ H_1: \text{there is relationship between LOC and Number of Defects.} \]

- The Spearman rank test rejects the null hypothesis and conclude that there is a weak but positive relationship between LOC and defects.
  - The spearman correlation is from 0.259 to 0.585.
  - Larger programs tend to have more defects.
Further studies show that a small number of programs account for a large number of defects

- For example, top 10% of the largest program account for about 46% Eclipse 3.0 defects,
- We could quickly locate a large number of defects by simply ranking the programs by metrics
- Termed “ranking ability” by Fenton et al. (2000)

<table>
<thead>
<tr>
<th></th>
<th>File Level</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Top 5%</td>
<td>Top 10%</td>
<td>Top 15%</td>
<td>Top 20%</td>
</tr>
<tr>
<td>Pre-release</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>defects</td>
<td>2.0</td>
<td>24.57%</td>
<td>37.01%</td>
<td>46.99%</td>
<td>53.48%</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>28.82%</td>
<td>43.46%</td>
<td>53.97%</td>
<td>61.01%</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>32.98%</td>
<td>46.28%</td>
<td>55.05%</td>
<td>62.29%</td>
</tr>
<tr>
<td>Post-release</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>defects</td>
<td>2.0</td>
<td>34.16%</td>
<td>46.87%</td>
<td>55.73%</td>
<td>61.88%</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>28.09%</td>
<td>40.52%</td>
<td>47.72%</td>
<td>54.31%</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>29.97%</td>
<td>44.05%</td>
<td>52.41%</td>
<td>60.62%</td>
</tr>
</tbody>
</table>
The Ranking Ability of LOC /2

- The cumulative distribution of defect can be visualized as an “Alberg diagram” like diagram.
- The modules are ordered by LOC.

Package level (pre-release)

File level (post-release)
Further analysis shows that the “ranking ability of LOC” can be modeled by a Weibull function

\[ P(x) = 1 - \exp\left(-\left(\frac{x}{\gamma}\right)^\beta\right) \]

(\(\gamma > 0, \beta > 0\))

<table>
<thead>
<tr>
<th></th>
<th>Eclipse</th>
<th>(\gamma)</th>
<th>(\beta)</th>
<th>(R^2)</th>
<th>(S_e)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre-release</strong> defects</td>
<td>2.0</td>
<td>0.259</td>
<td>0.897</td>
<td>0.991</td>
<td>0.023</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>0.207</td>
<td>0.830</td>
<td>0.995</td>
<td>0.017</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>0.193</td>
<td>0.780</td>
<td>0.992</td>
<td>0.019</td>
</tr>
<tr>
<td><strong>Post-release</strong> defects</td>
<td>2.0</td>
<td>0.190</td>
<td>0.811</td>
<td>0.986</td>
<td>0.026</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>0.242</td>
<td>0.853</td>
<td>0.988</td>
<td>0.026</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>0.203</td>
<td>0.827</td>
<td>0.993</td>
<td>0.019</td>
</tr>
</tbody>
</table>
The Implications of the Empirical Results

- The regularity of LOC distribution implies that:
  - a small percentage of programs have high complexity (*small program phenomenon*).
  - The *small program phenomenon* shows that in practices, programmers do not adhere to complexity thresholds strictly.
  - A small percentage of most complex programs are responsible for a large number of defects, while a large number of less complex programs contain a small number of defects.
  - By using LOC, we can quickly locate a large number of defects.
To explore the LOC’s ability in defect prediction, we examine the defect density of the top $k\%$ largest programs ($dd_k\%$):

$$dd_k\% = \frac{\text{(the number of defects the top } k\% \text{ largest modules contain)}}{\text{(the total KLOC of top } k\% \text{ largest modules)}} \times 100\%$$
Predicting defect counts /2

- As an example, we calculate the defect density values from the 10% largest Eclipse 3.0 programs, and then use the obtained values to predict the number of total defects in the system.

<table>
<thead>
<tr>
<th></th>
<th>File Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Defect Density</td>
</tr>
<tr>
<td>Pre-release defects</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td>Post-release defects</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
</tr>
</tbody>
</table>
Many attributes (such as complexity metrics, AST metrics, etc.) are used in existing defect prediction models.

We propose a LOC-based method for predicting defective components.

Classification models are built.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Classifier in WEKA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>Logistic</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>NaiveBayes</td>
</tr>
<tr>
<td>Decision Tree (C4.5)</td>
<td>J48</td>
</tr>
<tr>
<td>K-Star</td>
<td>KStar</td>
</tr>
</tbody>
</table>
Before training the prediction models, we firstly use logarithmic filter to transform data $n$ into their natural logarithms $\ln(n)$.
  - the transformation makes the data range narrower and make it easy for classifiers to learn.

We then construct the classification models using LOC data.

We use the 10-fold cross-validation to evaluate classification models.
Classifying Defective Components /3

To evaluate the predication model, we use Recall, Precision, F-measure, and Accuracy:

\[
\text{Recall} = \frac{TP}{TP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}
\]

\[
F\text{-}\text{measure} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}
\]

\[
\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN}
\]

The values of Recall, Precision, F-measure and Accuracy are between 0 and 1, the higher the better.
The cross-validation results for Eclipse 3.0 dataset:

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Pre-release</th>
<th>Post-release</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall (%)</td>
<td>Precision (%)</td>
</tr>
<tr>
<td>Multilayer Perceptron</td>
<td>85.5%</td>
<td>72.0%</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>86.7%</td>
<td>72.0%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>89.4%</td>
<td>71.2%</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>88.9%</td>
<td>71.7%</td>
</tr>
<tr>
<td>K-Star</td>
<td>87.5%</td>
<td>71.6%</td>
</tr>
</tbody>
</table>
Classifying Defective Components /5

- For the Eclipse dataset, all classification models obtain good results:
  - For pre-release results:
    - Recall values are above 85%
    - Precision values are above 71%
    - F-measures are about 0.79
    - Acc values are about 70%
  - For post-release results:
    - Recall ranging from 67% to 77%
    - Precision ranging from 63% to 68%
    - F-measure and Acc values are about 70%
Replication Study on NASA dataset

- As a replication study, we experiment with the NASA IV&V Facility Metrics Data Program (MDP) repository.
  - The data is collected from many NASA projects such as flight control, spacecraft instrument, storage management, and scientific data processing.
  - Developed in C/C++/Java
  - Very different from the Eclipse system.
- The NASA datasets contain software measurement data and associated defect data.
Results of Replication Study

- The results of the replication study are confirmative:
  - The distribution of LOC and defects are highly skewed.
  - There is a weak but positive relationship between LOC and defects.
  - The distribution of defects is Weibull distribution when modules are ranked by LOC ($R^2$ from 0.948 to 0.998).
  - A small percentage of the largest modules (e.g., top 10%) contain a large percentage of defects (e.g., 51% - 100%).
  - We can predict the defect counts based on defect density (e.g., MMRE = 16.94% based on the top 10% largest modules).
  - We can predict defect-prone components based on LOC (with Recall 90.1%, Precision 60.3% and F-measure 0.72).
Conclusion

- We investigated the relationship between LOC and Defects.
- Our experiments show that simple static code attributes such as LOC can be useful indicators of software quality.

Future work:
- Further analysis of LOC-Defect relationship
- Cross-project defect prediction
  - Can we use a model built from one project for a new project?
  - Within Company/Cross Company prediction
Thank you!

Hongyu Zhang
Associate Professor
School of Software, Tsinghua University
Beijing 100084, China
Email: hongyu@tsinghua.edu.cn
Web: http://thss.tsinghua.edu.cn/hongyu.htm