Defect Prediction on a Legacy Industrial Software: A Case Study on Software with Few Defects

Yavuz Koroglu¹  Alper Sen¹  Doruk Kutluay²
Akin Bayraktar²  Yalcin Tosun²  Murat Cinar²  Hasan Kaya²

¹Department of Computer Engineering
Bogazici University, Turkey
yavuz.koroglu@boun.edu.tr
depend.cmpe.boun.edu.tr

²NETAS Telecommunications
Istanbul, Turkey

1 Introduction

2 Methodology
   - Metrics
   - Predictive Models
   - Fine Tuning
   - Defect Prediction

3 Discussion
   - Threats to Validity
   - Conclusion

4 References

5 Appendix
Motivation

- Most software is shipped with **defects**.
- **Test** the software to detect defects.
- **Scalability** of testing is an issue.
  - According to a study in 2005, 79% of Microsoft developers are dedicated to writing unit tests [10].

80:20 Rule

- 80% of defects reside in the 20% of the software.
- Can we predict defective parts of the software to direct the testing effort?
Proposed Approach

- Use several Machine Learning (ML) techniques used in literature.
  - Naive Bayes [11], J48 Decision Tree [8], Random Forest [4, 9], Logistic Regression [7], Ensemble methods etc.
- **Predict** defective files.
- Direct testing effort **defect-prone** files.
NETAS

- **#1** systems integration company in Turkey.
- Offers networking, security, cloud, communication, maintainence, defense, public safety and e-government solutions.
- **First R&D** company in Turkey (founded in 1967).
Experius Project

- A multimedia app server project for VoIP communications.
- Mainly written in Java.
- Maintained via,
  - Issue tracking tool JIRA and
  - Version control system ClearCase.
- Large (~ 35K Java .class files).
- Low defect density (4%).
Methodology

1. Collect Metrics
2. Learn Predictive Models
3. Fine Tune Metrics/Models
4. Predict Defects

Collect Metrics

Learn Predictive Models

Fine Tune Metrics/Models

Predict Defects
Each file of each version is tagged as **defective** or **non-defective**.

Each entry contains metrics collected from current and previous versions of the file.

Training set for version 11.2 contains 144111 entries in total where 5923 entries (sum of previous versions) contain defects.

<table>
<thead>
<tr>
<th>Version</th>
<th># Files</th>
<th># Defective</th>
<th>% Defective</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>31758</td>
<td>1584</td>
<td>5%</td>
</tr>
<tr>
<td>10.1</td>
<td>32600</td>
<td>1725</td>
<td>5%</td>
</tr>
<tr>
<td>10.2</td>
<td>33332</td>
<td>1273</td>
<td>4%</td>
</tr>
<tr>
<td>10.3</td>
<td>34702</td>
<td>1920</td>
<td>5%</td>
</tr>
<tr>
<td>10.4</td>
<td>37554</td>
<td>1005</td>
<td>3%</td>
</tr>
<tr>
<td>11.2</td>
<td>37988</td>
<td>1295</td>
<td>3%</td>
</tr>
</tbody>
</table>
### Definition

- Measures of a Java `.class` file.
- Related to the **defect-proneness** of the `.class` file.

### Types of Metrics

1. **Product Metrics:** Collected from the `.class` file.
2. **Process Metrics:** Collected from previous versions of the file via JIRA/ClearCase.
# Product Metrics

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WMC</td>
<td>Weighted Method Count</td>
</tr>
<tr>
<td>2</td>
<td>DIT</td>
<td>Depth of Inheritance Tree</td>
</tr>
<tr>
<td>3</td>
<td>NOC</td>
<td>Number of Children</td>
</tr>
<tr>
<td>4</td>
<td>CBO</td>
<td>Coupling Between Objects</td>
</tr>
<tr>
<td>5</td>
<td>RFC</td>
<td>Response for Class</td>
</tr>
<tr>
<td>6</td>
<td>LCOM</td>
<td>Lack of Cohesion in Methods</td>
</tr>
<tr>
<td>7</td>
<td>Ca</td>
<td>Afferent Couplings</td>
</tr>
<tr>
<td>8</td>
<td>Ce</td>
<td>Efferent Couplings</td>
</tr>
<tr>
<td>9</td>
<td>NPM</td>
<td>Number of Public Methods</td>
</tr>
<tr>
<td>10</td>
<td>LCOM3</td>
<td>Lack of Cohesion in Methods</td>
</tr>
<tr>
<td>11</td>
<td>LOC</td>
<td>Lines of Code</td>
</tr>
<tr>
<td>12</td>
<td>DAM</td>
<td>Data Access Metric</td>
</tr>
<tr>
<td>13</td>
<td>MOA</td>
<td>Measure of Aggregation</td>
</tr>
<tr>
<td>14</td>
<td>MFA</td>
<td>Measure of Functional Abstraction</td>
</tr>
<tr>
<td>15</td>
<td>CAM</td>
<td>Cohesion Among Methods of Class</td>
</tr>
<tr>
<td>16</td>
<td>IC</td>
<td>Inheritance Coupling</td>
</tr>
<tr>
<td>17</td>
<td>CBM</td>
<td>Coupling Between Methods</td>
</tr>
<tr>
<td>18</td>
<td>AMC</td>
<td>Average Method Complexity</td>
</tr>
</tbody>
</table>

- Product metrics are collected via CKJM Extended [3].
- Metrics are collected from binary (.class files).
## Process Metrics

### Process Metrics [7]

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>NDPV</td>
<td># Defects in the Previous Version</td>
</tr>
<tr>
<td>20</td>
<td>NML</td>
<td># Modified Lines</td>
</tr>
<tr>
<td>21</td>
<td>NDC</td>
<td># Distinct Commiters</td>
</tr>
</tbody>
</table>

### Additional process metrics

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>PBC</td>
<td>Previous version Bug Criticality(1-5)</td>
</tr>
<tr>
<td>23</td>
<td>ABC</td>
<td>Average Bug Criticality(1-5)</td>
</tr>
<tr>
<td>24</td>
<td>PBF</td>
<td>Previous version Bug Fixes</td>
</tr>
<tr>
<td>25</td>
<td>ABF</td>
<td>Average Bug Fixes</td>
</tr>
</tbody>
</table>
25 metrics for each version of each file.

Defective entries are rare. Therefore we used SMOTE (Synthetic Minority Oversampling TEchnique) to oversample the defects [1].

- SMOTE-Random Forest is known to work well with imbalanced data [2].

We increase the number of defective entries by 20x to have approximately equal number of instances for each class.
1 Introduction

2 Methodology
   - Metrics
   - Predictive Models
   - Fine Tuning
   - Defect Prediction

3 Discussion
   - Threats to Validity
   - Conclusion

4 References

5 Appendix
Candidate Predictive Models

Candidates

- Random Forest
- Logistic Regression
- J48 Decision Tree
- Naive Bayes

Candidate Selection Criteria

- Found in literature and
- Model training time should be small (an hour).
Training

- Training is done using **WEKA** [5].
- Changed **hyperparameters** to optimize.
- Used **different oversampling ratios** to generate Receiver Operating Characteristic (ROC) curve.
  - Area Under ROC Curve (AUC) is a measure of **predictive power**.
  - We use AUC to choose best model.
Random Forest + Logistic Regression has the best predictive power. Random Forest is faster with second best predictive power.
1 Introduction

2 Methodology
   ■ Metrics
   ■ Predictive Models
   ■ Fine Tuning
   ■ Defect Prediction

3 Discussion
   ■ Threats to Validity
   ■ Conclusion

4 References

5 Appendix
Measuring Predictive Power

Confusion Matrix

<table>
<thead>
<tr>
<th>actual</th>
<th>predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>(f_n)</td>
</tr>
</tbody>
</table>

- 1: defective, 0: non-defective
- Recall: \( \frac{t_p}{f_n + t_p} \)
- Precision: \( \frac{t_p}{f_p + t_p} \)
- Accuracy: \( \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \)
- Prevalence: \( \frac{t_p + f_p}{t_p + t_n + f_p + f_n} \)

Details

- Recall is a **measure of completeness**.
- Precision is a **measure of quality**.
- Accuracy is a **measure of predictive power**.
- Prevalence is a **measure of size**.
- **Tradeoff** between parameters.
- The company chose from several options we provided according to their needs.
1. Introduction

2. Methodology
   - Metrics
   - Predictive Models
   - Fine Tuning
   - Defect Prediction

3. Discussion
   - Threats to Validity
   - Conclusion

4. References

5. Appendix
Confusion Matrix

<table>
<thead>
<tr>
<th>actual</th>
<th>predicted</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34242</td>
<td>2451</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>612</td>
<td>683</td>
<td></td>
</tr>
</tbody>
</table>

- **1**: defective, **0**: non-defective
- **Recall**: 0.527
- **Precision**: 0.218
- **Accuracy**: 0.919
- **Prevalence**: 0.087

- **Prevalence**: 8.7% of the Java files are marked as defect-prone.
- **Precision**: 21.8% of the defect-prone files contain actual defects.
- **Recall**: The 8.7% marked as defect-prone contains 52.7% of all defects.
- As a last note, our model has a **high accuracy** (91.9%).
Related Work

Malhotra [8]

- Decision Trees and Random Forests trained on multiple projects.
- AUC values;
  - Between 0.66 and 1 for Random Forests.
  - Our AUC is 0.73.
  - Our model performs better compared to software with low defect rate (below 5%).

Hall [6]

- Multiple models.
- Best Recall values on file level range from 0.40 and 0.65. Our recall (0.527) is close to their mean.
### Related Work

**Gothra [4]**

- Multiple models.
- Our best AUC (0.75) is better than 5/10 and comparable to 3/10 projects.

**Tosun et al. [11]**

- Study on Turkish industry, uses Naive Bayes.
- Exploits **undersampling**. Undersampling in our case → Small training set.
- Defect rate of the underlying software is higher (Software with up to 18% defectives).
## Impact of Additional Metrics

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>PBC</td>
<td>Previous version Bug Criticality(1-5)</td>
</tr>
<tr>
<td>23</td>
<td>ABC</td>
<td>Average Bug Criticality(1-5)</td>
</tr>
<tr>
<td>24</td>
<td>PBF</td>
<td>Previous Bug Fixes</td>
</tr>
<tr>
<td>25</td>
<td>ABF</td>
<td>Average Bug Fixes</td>
</tr>
</tbody>
</table>
Impact of Additional Metrics

Intuition

- Defect related metrics → positive impact on predictive power.
- Bug information for each version of each class file was readily available.
- Bug information must be related with defect-proneness.

Approach

- Use **feature selection techniques** to rate the relevance of additional metrics.
- Train same model **with and without** additional metrics.
## Metric Ranking

Top 10 metrics according to their individual information gain.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NDC</td>
<td>Number of Distinct Commiters</td>
</tr>
<tr>
<td>2</td>
<td>NDPV</td>
<td>Number of Defects in Previous Version</td>
</tr>
<tr>
<td>3</td>
<td>PBF</td>
<td>Previous version Bug Fixes</td>
</tr>
<tr>
<td>4</td>
<td>NML</td>
<td>Number of Modified Lines</td>
</tr>
<tr>
<td>5</td>
<td>PBC</td>
<td>Previous version Bug Criticality</td>
</tr>
<tr>
<td>6</td>
<td>ABF</td>
<td>Average Bug Fixes</td>
</tr>
<tr>
<td>7</td>
<td>ABC</td>
<td>Average Bug Criticality</td>
</tr>
<tr>
<td>8</td>
<td>DIT</td>
<td>Depth in Inheritance Tree</td>
</tr>
<tr>
<td>9</td>
<td>Ca</td>
<td>Afferent Couplings</td>
</tr>
<tr>
<td>10</td>
<td>MOA</td>
<td>Measure of Aggregation</td>
</tr>
</tbody>
</table>

Top 7 metrics are **process metrics**.
### Additional Metrics Cont’d

#### With Additional Metrics

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall (R)</td>
<td>0.527</td>
</tr>
<tr>
<td>Precision (P)</td>
<td>0.218</td>
</tr>
<tr>
<td>Accuracy (A)</td>
<td>0.919</td>
</tr>
<tr>
<td>Positive Prevalence</td>
<td>0.087</td>
</tr>
</tbody>
</table>

#### Without Additional Metrics

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall (R)</td>
<td>0.444</td>
</tr>
<tr>
<td>Precision (P)</td>
<td>0.165</td>
</tr>
<tr>
<td>Accuracy (A)</td>
<td>0.905</td>
</tr>
<tr>
<td>Positive Prevalence</td>
<td>0.090</td>
</tr>
</tbody>
</table>
1 Introduction

2 Methodology
- Metrics
- Predictive Models
- Fine Tuning
- Defect Prediction

3 Discussion
- Threats to Validity
- Conclusion

4 References

5 Appendix
Threats to Validity

1. Inaccurate Metric Collection
2. Missing History due to Unhandled Refactorings
3. Result Granularity Issues
4. External Validity Issues
Inaccurate Metric Collection

**Lines of Code (LOC) Problem**
- LOC metric counts the lines in binary (not source).
- Correlation between binary LOC and source LOC.
- Binary LOC is related to defect-proneness.

**Number of Modified Lines (NML) Problem**
- We weren’t provided with NML information.
- We approximated the value as the difference between LOCs.
Missing History due to Unhandled Refactorings

Scenario

1. Java .class file X gets renamed to Y in new version.
2. Our metric extraction script **CANNOT** find history of Y.

Problem Severity

- 16094 of 144111 Java files have no history.
- Some of the files are genuinely new, some are not.
- No way to distinguish such files.
Result Granularity and External Validity Issues

Result Granularity

- The percentage of defect-prone files **DOES NOT** represent percentage of defect-prone LOC.
- We believe that our results sufficiently approximates the percentage of the project.

External Validity

- **DO NOT** claim that Random Forest with 25 metrics should achieve the same predictive power in other software.
- However, our results are similar to several related work [4, 8].
1 Introduction

2 Methodology
   - Metrics
   - Predictive Models
   - Fine Tuning
   - Defect Prediction

3 Discussion
   - Threats to Validity
   - Conclusion

4 References

5 Appendix
Conclusions

- We successfully predicted defects of an industrial software project.
- Our method successfully predicts 52.7% of all defects by suggesting 8.7% of the files with 91.9% overall accuracy.
- We can customize the model according to needs (get more recall at the cost of overall accuracy).
- Feedback from the company indicates that we achieve similar predictive performance on version 12.0.
- Our model is ready to be integrated into the company’s Continuous Integration (CI) pipeline.
- In the future, we aim to train our model on multiple projects.
Thank You.


Predict Bugs in Version 11.2 via Random Forest

| -I100 -K5 -D3 |  
| 0 | 34045 | 2648 |  
| 1 | 599 | 696 |  

- $R = 0.537$  
- $P = 0.208$  
- $A = 0.915$

| -I100 -K5 -D3 -C |  
| 0 | 30166 | 6527 |  
| 1 | 465 | 830 |  

- $R = 0.641$  
- $P = 0.113$  
- $A = 0.82$

| -I100 -K8 -D5 |  
| 0 | 34242 | 2451 |  
| 1 | 612 | 683 |  

- $R = 0.527$  
- $P = 0.218$  
- $A = 0.919$

| -I100 -K8 -D5 -C |  
| 0 | 33226 | 3467 |  
| 1 | 664 | 631 |  

- $R = 0.487$  
- $P = 0.154$  
- $A = 0.885$
**Predict Bugs in Version 11.2 via J48 Decision Tree**

### -C 0.1

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34138</td>
<td>2555</td>
</tr>
<tr>
<td>1</td>
<td>755</td>
<td>540</td>
</tr>
</tbody>
</table>

\[ R = 0.417 \]
\[ P = 0.174 \]
\[ A = 0.913 \]

### -C 0.25

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34137</td>
<td>2556</td>
</tr>
<tr>
<td>1</td>
<td>756</td>
<td>539</td>
</tr>
</tbody>
</table>

\[ R = 0.416 \]
\[ P = 0.174 \]
\[ A = 0.913 \]

### -C 0.25 -R

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33928</td>
<td>2765</td>
</tr>
<tr>
<td>1</td>
<td>782</td>
<td>513</td>
</tr>
</tbody>
</table>

\[ R = 0.396 \]
\[ P = 0.156 \]
\[ A = 0.907 \]

### -C 0.1 -R

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33899</td>
<td>2794</td>
</tr>
<tr>
<td>1</td>
<td>791</td>
<td>504</td>
</tr>
</tbody>
</table>

\[ R = 0.389 \]
\[ P = 0.153 \]
\[ A = 0.906 \]
Predict Bugs in Version 11.2 via Naive Bayes

<table>
<thead>
<tr>
<th>Standard</th>
<th>High Recall</th>
<th>Highest Recall</th>
<th>High Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Table" /></td>
<td><img src="image2.png" alt="Table" /></td>
<td><img src="image3.png" alt="Table" /></td>
<td><img src="image4.png" alt="Table" /></td>
</tr>
</tbody>
</table>

- **Standard**
  - $R = 0.432$
  - $P = 0.208$
  - $A = 0.924$

- **High Recall**
  - $R = 0.494$
  - $P = 0.182$
  - $A = 0.907$

- **Highest Recall**
  - $R = 0.546$
  - $P = 0.141$
  - $A = 0.871$

- **High Precision**
  - $R = 0.372$
  - $P = 0.236$
  - $A = 0.937$
Random Forest with 100 trees of 5 attributes and 3 depth has been used.

Version 10.3 has a more reliable version history.

Version 10.2 is not predictable because there is not enough history to learn a good model for it.