

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

Yavuz Koroglu<sup>1</sup>    Alper Sen<sup>1</sup>    Doruk Kutluay<sup>2</sup>  
Akin Bayraktar<sup>2</sup>    Yalcin Tosun<sup>2</sup>    Murat Cinar<sup>2</sup>    Hasan Kaya<sup>2</sup>

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

<sup>2</sup>NETAS Telecommunications  
Istanbul, Turkey

Fourth Intl. Workshop on Conducting Empirical Studies in  
Industry CESI 2016 - An ICSE 2016 Workshop

## 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

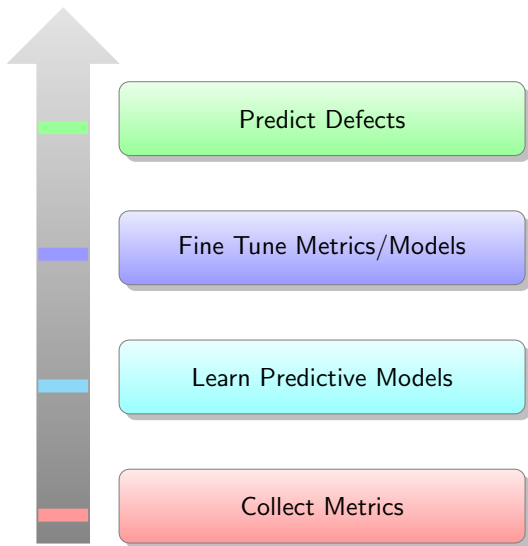
- **#1** systems integration company in Turkey.
- Offers networking, security, cloud, communication, maintenance, defense, public safety and e-government solutions.
- **First R&D** company in Turkey (founded in 1967).

# Legacy Software Description

## 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 Introduction

## 2 Methodology

### ■ Metrics

- Predictive Models
- Fine Tuning
- Defect Prediction

## 3 Discussion

- Threats to Validity
- Conclusion

## 4 References

## 5 Appendix



# Data

- 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.

## Collected Data

Version	# Files	# Defective	% Defective
10.0	31758	1584	5%
10.1	32600	1725	5%
10.2	33332	1273	4%
10.3	34702	1920	5%
10.4	37554	1005	3%
11.2	37988	1295	3%

# Collected Metrics

## 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

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

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

# Process Metrics

## Process Metrics [7]

#	Metric	Description
19	NDPV	# Defects in the Previous Version
20	NML	# Modified Lines
21	NDC	# Distinct Committers

## Additional process metrics

#	Metric	Description
22	PBC	Previous version Bug Criticality(1-5)
23	ABC	Average Bug Criticality(1-5)
24	PBF	Previous version Bug Fixes
25	ABF	Average Bug Fixes

- 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

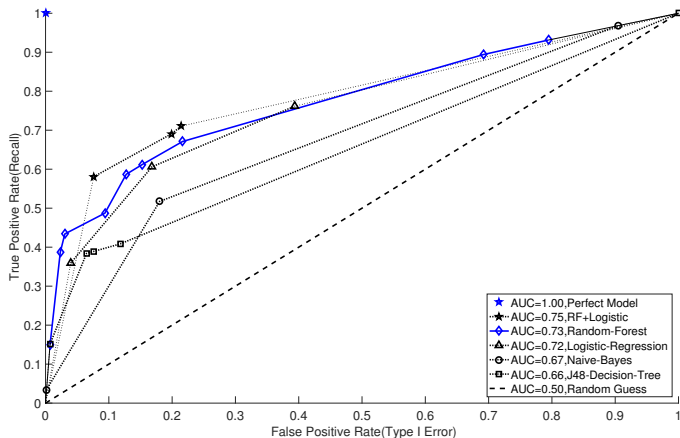
## Candidate Selection Criteria

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

- Training 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.



# Model Comparison



- 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

		predicted	
		0	1
actual	0	$t_n$	$f_p$
	1	$f_n$	$t_p$

- **1:** defective, **0:** non-defective
- **Recall:**  $t_p / (f_n + t_p)$
- **Precision:**  $t_p / (f_p + t_p)$
- **Accuracy:**  
 $(t_p + t_n) / (t_p + t_n + f_p + f_n)$
- **Prevalence:**  
 $(t_p + f_p) / (t_p + t_n + f_p + f_n)$

## 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

# Predict Defects in Version 11.2 via Random Forest

Confusion Matrix

		predicted	
		0	1
actual	0	34242	2451
	1	612	683

- **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%).

## 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.

## 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

#	Metric	Description
22	PBC	Previous version Bug Criticality(1-5)
23	ABC	Average Bug Criticality(1-5)
24	PBF	Previous Bug Fixes
25	ABF	Average Bug Fixes



# 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.

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

Top 7 metrics are **process metrics**.

## Additional Metrics Cont'd

### With Additional Metrics

		predicted	
		0	1
actual	0	34242	2451
	1	612	683

Measure	Value
Recall ( $R$ )	0.527
Precision ( $P$ )	0.218
Accuracy ( $A$ )	0.919
Positive Prevalence	0.087

### Without Additional Metrics

		predicted	
		0	1
actual	0	33793	2900
	1	720	575

Measure	Value
Recall ( $R$ )	0.444
Precision ( $P$ )	0.165
Accuracy ( $A$ )	0.905
Positive Prevalence	0.090

## 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 **CAN NOT** 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

- We **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.

# References I

- [1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer.  
Smote: Synthetic minority over-sampling technique.  
*J. Artif. Int. Res.*, 16(1):321–357, June 2002.
- [2] Chao Chen, Andy Liaw, and Leo Breiman.  
Using Random Forest to Learn Imbalanced Data.  
Technical report, Department of Statistics, University of Berkeley, 2004.
- [3] CKJM Extended, [http://gromit.iar.pwr.wroc.pl/p\\_inf/ckjm/](http://gromit.iar.pwr.wroc.pl/p_inf/ckjm/).

## References II

- [4] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. *In Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE '15*, 2015.
- [5] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.
- [6] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans. Softw. Eng.*, 38(6):1276–1304, November 2012.

## References III

- [7] Lech Madeyski and Marian Jureczko.  
Which process metrics can significantly improve defect prediction models? an empirical study.  
*Software Quality Journal*, 23(3):393–422, September 2015.
- [8] Ruchika Malhotra.  
A systematic review of machine learning techniques for software fault prediction.  
*Appl. Soft Comput.*, 27(C):504–518, February 2015.
- [9] Julie Moeyersoms, Enric Junqué de Fortuny, Karel Dejaeger, Bart Baesens, and David Martens.  
Comprehensible software fault and effort prediction: A data mining approach.  
*Journal of Systems and Software*, 100:80–90, 2015.

## References IV

- [10] Xiao Qu and Brian Robinson.  
A case study of concolic testing tools and their limitations.  
In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement, ESEM '11*, 2011.
- [11] Ayşe Tosun, Ayşe Bener, Burak Turhan, and Tim Menzies.  
Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry.  
*Inf. Softw. Technol.*, 52(11):1242–1257, November 2010.

# Predict Bugs in Version 11.2 via Random Forest

-I100 -K5 -D3

	0	1
0	34045	2648
1	599	696

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

-I100 -K5 -D3 -C

	0	1
0	30166	6527
1	465	830

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

-I100 -K8 -D5

	0	1
0	34242	2451
1	612	683

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

-I100 -K8 -D5 -C

	0	1
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

	0	1
0	34138	2555
1	755	540

$R = 0.417$   
 $P = 0.174$   
 $A = 0.913$

-C 0.25

	0	1
0	34137	2556
1	756	539

$R = 0.416$   
 $P = 0.174$   
 $A = 0.913$

-C 0.25 -R

	0	1
0	33928	2765
1	782	513

$R = 0.396$   
 $P = 0.156$   
 $A = 0.907$

-C 0.1 -R

	0	1
0	33899	2794
1	791	504

$R = 0.389$   
 $P = 0.153$   
 $A = 0.906$

# Predict Bugs in Version 11.2 via Naive Bayes

## Standard

	0	1
0	34557	2136
1	735	560

$R = 0.432$   
 $P = 0.208$   
 $A = 0.924$

## High Recall

	0	1
0	33822	2871
1	655	640

$R = 0.494$   
 $P = 0.182$   
 $A = 0.907$

## Highest Recall

	0	1
0	32383	4310
1	588	707

$R = 0.546$   
 $P = 0.141$   
 $A = 0.871$

## High Precision

	0	1
0	35136	1557
1	813	482

$R = 0.372$   
 $P = 0.236$   
 $A = 0.937$

# Versions 10.3 and 10.4

10.3

	0	1
0	31191	1591
1	1108	812

$R = 0.423$

$P = 0.338$

$A = 0.92$

10.4

	0	1
0	31708	4741
1	513	492

$R = 0.489$

$P = 0.094$

$A = 0.86$

- 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.