Replication of Defect Prediction Studies

Problems, Pitfalls and Recommendations

Thilo Mende

University of Bremen, Germany

PROMISE'10
12.09.2010

Replication is a waste of time.

"'Replicability is not Reproducibility: Nor is it Good Science"'
–Drummond (2009)

Replication is a waste of time.

"‘Replicability is not Reproducibility: Nor is it Good Science"’

–Drummond (2009)

Reproduction	Replication
small changes	no changes
desirable	wasted effort

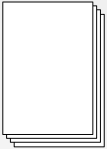
Replication is a waste of time.

"'Replicability is not Reproducibility: Nor is it Good Science"'
–Drummond (2009)

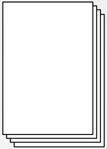
Reproduction	Replication
small changes	no changes
desirable	wasted effort

Replication $\rightarrow$ **identical** results

So why did I do this?

 +  = ?

So why did I do this?

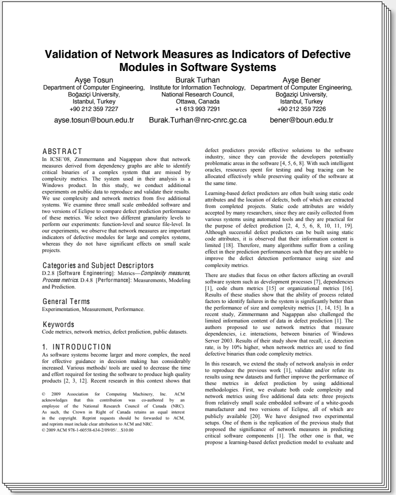
 +  = ?

Prerequisite for:

- Reproducing results on other data sets
- Further analyses

Case Study 1

- Tosun et al.: "‘Validation of network measures as indicators of defective modules in software systems”, PROMISE 2009.
- Reproduction of Zimmermann and Nagappan (2008)



Validation of Network Measures as Indicators of Defective Modules in Software Systems

Ayşe Tosun
Department of Computer Engineering,
Boğaziçi University,
Istanbul, Turkey
+90 212 359 7227
ayse.tosun@boun.edu.tr

Burak Turhan
Institute for Information Technology,
National Research Council,
Ottawa, Canada
+1 613 993 7291
Burak.Turhan@nrc-cnrc.gc.ca

Ayşe Bener
Department of Computer Engineering,
Boğaziçi University,
Istanbul, Turkey
+90 212 359 7226
bener@boun.edu.tr

ABSTRACT
In ICSE'08, Zimmermann and Nagappan show that network measures derived from dependency graphs are able to identify critical binaries of a complex system that are missed by complexity metrics. The system used in their analysis is a Windows product. In this study, we conduct additional experiments on public data to reproduce and validate their results. We use complexity and network metrics from five additional systems. We examine three small scale embedded software and two versions of Eclipse to compare defect prediction performance of these metrics. We select two different granularity levels to perform our experiments: function-level and source file-level. In our experiments, we observe that network measures are important indicators of defective modules for large and complex systems, whereas they do not have significant effects on small scale projects.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics—*Complexity measures, Process metrics*. D.4.8 [Performance]: Measurements, Modeling and Prediction.

General Terms
Experimentation, Measurement, Performance.

Keywords
Code metrics, network metrics, defect prediction, public datasets.

1. INTRODUCTION
As software systems become larger and more complex, the need for effective guidance in decision making has considerably increased. Various methods/ tools are used to decrease the time and effort required for testing the software to produce high quality products [2, 3, 12]. Recent research in this context shows that defect predictors provide effective solutions to the software industry, since they can provide the developers potentially problematic areas in the software [4, 5, 6, 8]. With such intelligent oracles, resources spent for testing and bug tracing can be allocated effectively while preserving quality of the software at the same time.

Learning-based defect predictors are often built using static code attributes and the location of defects, both of which are extracted from completed projects. Static code attributes are widely accepted by many researchers, since they are easily collected from various systems using automated tools and they are practical for the purpose of defect prediction [2, 4, 5, 6, 8, 10, 11, 19]. Although successful defect predictors can be built using static code attributes, it is observed that their information content is limited [18]. Therefore, many algorithms suffer from a ceiling effect in their prediction performances such that they are unable to improve the defect detection performance using size and complexity metrics.

There are studies that focus on other factors affecting an overall software system such as development processes [7], dependencies [1], code churn metrics [15] or organizational metrics [16]. Results of these studies show that the ability of process related factors to identify failures in the system is significantly better than the performance of size and complexity metrics [1, 14, 15]. In a recent study, Zimmermann and Nagappan also challenged the limited information content of data in defect prediction [1]. The authors proposed to use network metrics that measure dependencies, i.e. interactions, between binaries of Windows Server 2003. Results of their study show that recall, i.e. detection rate, is by 10% higher, when network metrics are used to find defective binaries than code complexity metrics.

In this research, we extend the study of network analysis in order to reproduce the previous work [1], validate and/or refute its results using new datasets and further improve the performance of these metrics in defect prediction by using additional methodologies. First, we evaluate both code complexity and network metrics using five additional data sets: three projects from relatively small scale embedded software of a white-goods manufacturer and two versions of Eclipse, all of which are publicly available [20]. We have designed two experimental setups. One of them is the replication of the previous study that proposed the significance of network measures in predicting critical software components [1]. The other one is that, we propose a learning-based defect prediction model to evaluate and

Data Sets

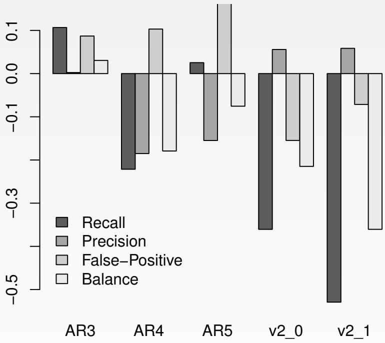
	AR 3-5	Eclipse
Input: Complexity	√	√
Input: Call-Graph	✗	✗
Dependent	√	?

Experimental Setup

	Logistic Regression	Naive Bayes
Transformation	—	√
Algorithm	√	√
Implementation	?	?
Cutoff	✗	✗
Evaluation	√	√
Performance Measures	√	√

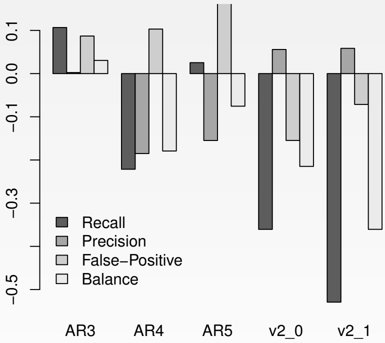
Results

Logistic Regression

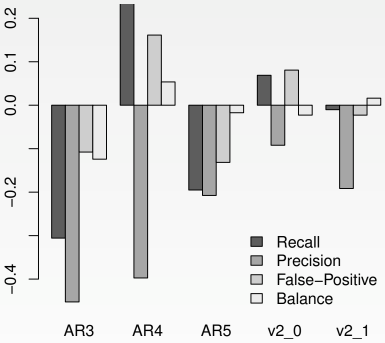


Results

Logistic Regression



Naive Bayes



Problems

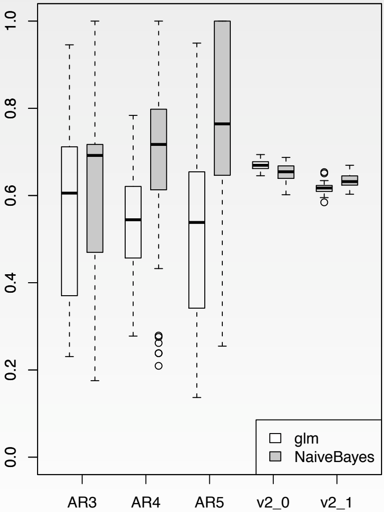
- Class-Labels
- Cutoff

Problems

- Class-Labels
- Cutoff
- Partitions with no positives
- Large variation

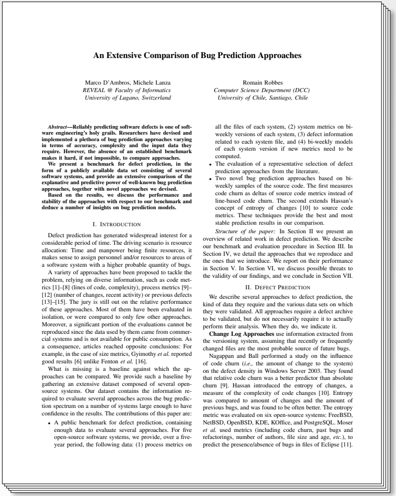
Problems

- Class-Labels
- Cutoff
- Partitions with no positives
- Large variation



(Non-Averaged *Balance*)

Case Study 2



An Extensive Comparison of Bug Prediction Approaches

Marco D'Ambros, Michele Lanza
REVEAL @ Faculty of Informatics
University of Lugano, Switzerland

Romain Robbes
Computer Science Department (DCC)
University of Chile, Santiago, Chile

Abstract—**Reliably predicting software defects is one of software engineering's holy grails. Researchers have devised and implemented a plethora of bug prediction approaches varying in terms of accuracy, complexity and the input data they require. However, the absence of an established benchmark makes it hard, if not impossible, to compare approaches.**

We present a benchmark for defect prediction, in the form of a publicly available data set consisting of several software systems, and provide an extensive comparison of the explanative and predictive power of well-known bug prediction approaches, together with novel approaches we devised.

Based on the results, we discuss the performance and stability of the approaches with respect to our benchmark and deduce a number of insights on bug prediction models.

I. INTRODUCTION

Defect prediction has generated widespread interest for a considerable period of time. The driving scenario is resource allocation: Time and manpower being finite resources, it makes sense to assign personnel and/or resources to areas of a software system with a higher probable quantity of bugs.

A variety of approaches have been proposed to tackle the problem, relying on diverse information, such as code metrics [1]–[8] (lines of code, complexity), process metrics [9]–[12] (number of changes, recent activity) or previous defects [13]–[15]. The jury is still out on the relative performance of these approaches. Most of them have been evaluated in isolation, or were compared to only few other approaches. Moreover, a significant portion of the evaluations cannot be reproduced since the data used by them came from commercial systems and is not available for public consumption. As a consequence, articles reached opposite conclusions: For example, in the case of size metrics, Gyimothy *et al.* reported good results [6] unlike Fenton *et al.* [16].

What is missing is a baseline against which the approaches can be compared. We provide such a baseline by gathering an extensive dataset composed of several open-source systems. Our dataset contains the information required to evaluate several approaches across the bug prediction spectrum on a number of systems large enough to have confidence in the results. The contributions of this paper are:

- A public benchmark for defect prediction, containing enough data to evaluate several approaches. For five open-source software systems, we provide, over a five-year period, the following data: (1) process metrics on all the files of each system, (2) system metrics on bi-weekly versions of each system, (3) defect information related to each system file, and (4) bi-weekly models of each system version if new metrics need to be computed.
- The evaluation of a representative selection of defect prediction approaches from the literature.
- Two novel bug prediction approaches based on bi-weekly samples of the source code. The first measures code churn as deltas of source code metrics instead of line-based code churn. The second extends Hassan's concept of entropy of changes [10] to source code metrics. These techniques provide the best and most stable prediction results in our comparison.

Structure of the paper: In Section II we present an overview of related work in defect prediction. We describe our benchmark and evaluation procedure in Section III. In Section IV, we detail the approaches that we reproduce and the ones that we introduce. We report on their performance in Section V. In Section VI, we discuss possible threats to the validity of our findings, and we conclude in Section VII.

II. DEFECT PREDICTION

We describe several approaches to defect prediction, the kind of data they require and the various data sets on which they were validated. All approaches require a defect archive to be validated, but do not necessarily require it to actually perform their analysis. When they do, we indicate it.

Change Log Approaches use information extracted from the versioning system, assuming that recently or frequently changed files are the most probable source of future bugs.

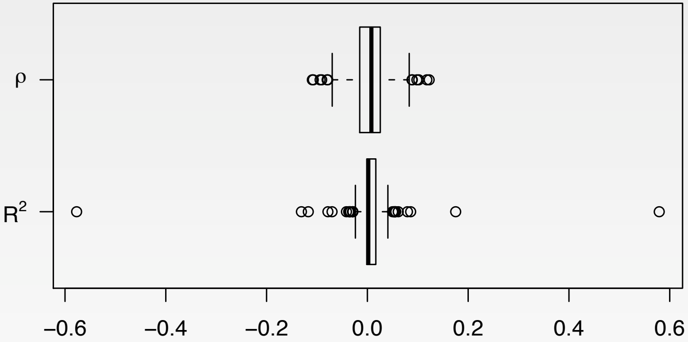
Nagappan and Ball performed a study on the influence of code churn (*i.e.*, the amount of change to the system) on the defect density in Windows Server 2003. They found that relative code churn was a better predictor than absolute churn [9]. Hassan introduced the entropy of changes, a measure of the complexity of code changes [10]. Entropy was compared to amount of changes and the amount of previous bugs, and was found to be often better. The entropy metric was evaluated on six open-source systems: FreeBSD, NetBSD, OpenBSD, KDE, KOffice, and PostgreSQL. Moser *et al.* used metrics (including code churn, past bugs and refactorings, number of authors, file size and age, *etc.*), to predict the presence/absence of bugs in files of Eclipse [11].

Marco D'Ambros et al.: "“An Extensive Comparison of Bug Prediction Approaches"”, MSR 2010.

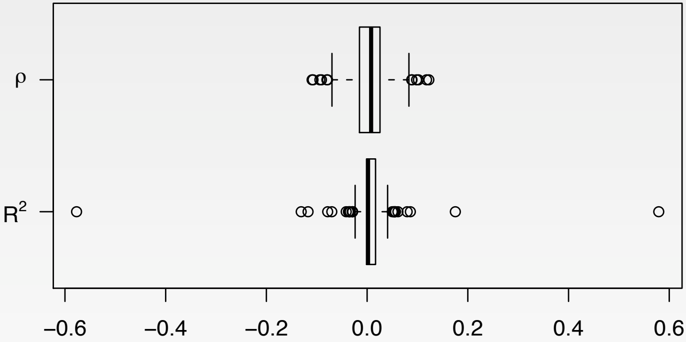
Setup

Mapping to data sets	$\surd$	5 open-source systems
Independent	$\surd$	25+6 sets
Dependent	$\surd$	# of defects
Algorithm	$\surd$	generalized linear models
Implementation	?	R's `glm`
Evaluation	$\surd$	50$\times$90:10 split
Performance Measures	$\surd$	Adjusted R^2, Spearman's ρ, custom scoring system

Results



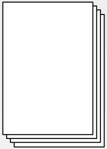
Results



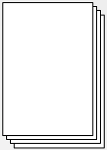
Scoring system

- 24 equal
- 5 rounding errors
- 2 outliers

The Summary

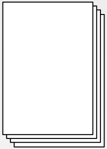
 +  =

The Summary

 +  = ?

Issues:

- Cutoffs
- Data transformations
- Some data sets too small → huge variance
- Implementation details
  - rounding
  - ties with spearman
  - …

 +  = ?

Issues:

- Cutoffs
- Data transformations
- Some data sets too small $\rightarrow$ huge variance
- Implementation details
  - rounding
  - ties with spearman
  - $\ldots$

Wasted Time?

The Summary

Issues:

- Cutoffs
- Data transformations
- Some data sets too small $\rightarrow$ huge variance
- Implementation details
  - rounding
  - ties with spearman
  - ...

Wasted Time? Maybe, but necessary prerequisite...

An Observation

Spearman correlation between **# input variables** and R^2:

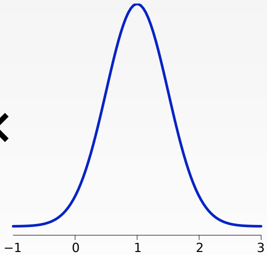
$$\rho = 0.890$$

An Observation

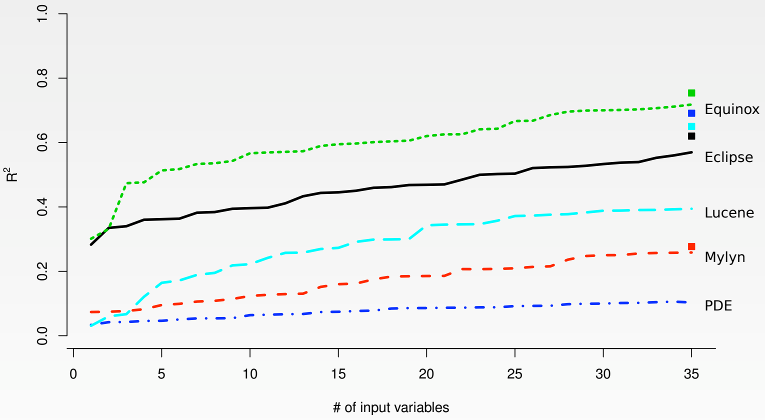
Spearman correlation between **# input variables** and R^2:

$$\rho = 0.890$$

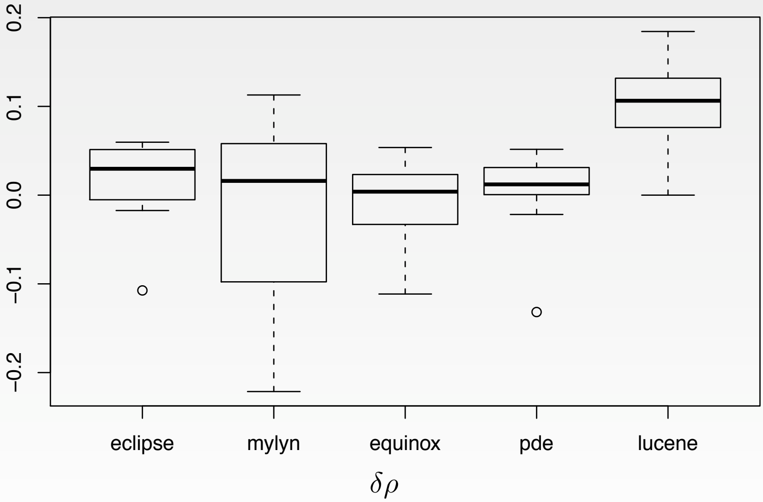
Generate **random** variables

LoC $\times$ 

Performance with Random Data



Compared to a Trivial Model



And now?

- Start sharing experiments?
- Or PROMISE repository on GitHub?
- …publishing 'official' results
- To improve best practices, e.g.
  - Minimal set of necessary information
  - Random and trivial models

Replication of Defect Prediction Studies
Problems, Pitfalls and Recommendations

Thilo Mende, University of Bremen
tmende@informatik.uni-bremen.de

Universität Bremen

Drummond, C. (2009). Replicability is not reproducibility: Nor is it good science. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning IV*.

Zimmermann, T. and N. Nagappan (2008). Predicting defects using network analysis on dependency graphs. In *International Conference on Software Engineering*, New York, NY, USA, pp. 531–540. ACM.