Towards Building a Universal Defect Prediction Model

by Feng Zhang, Iman Keivanloo, Audris Mockus & Ying Zou

A Paper Review Presentation

By
Ekaba Bisong
MSc Candidate
School of Computer Science
Carleton University
Nov. 10th 2016

Background

- A software **defect** is an **error**, **flaw**, **bug**, mistake, failure, or fault **in a computer program** or system **that** may **generate** an **inaccurate or unexpected outcome**, or precludes the software from behaving as intended.[1]

1. Rawat & Dubey, 2002. Software Defect Prediction Models for Quality Improvement: A Literature Study. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2
http://www.ijcsi.org/papers/IJCSI-9-5-2-288-296.pdf

- **Defect prediction** is extremely **essential in** the field of **software quality** and **software reliability**.[1]

- Defect prediction is comparatively **a novel research area** of software quality engineering.[1]

1. Rawat & Dubey, 2002. Software Defect Prediction Models for Quality Improvement: A Literature Study. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2 http://www.ijcsi.org/papers/IJCSI-9-5-2-288-296.pdf

Goal of Paper

- This paper **proposes a defect prediction model** that is **not constrained to** predict defects in **a single software project**, **but** rather that can be used **to predict defects across a variety of software projects**.

- A universal defect prediction model **would relieve the need for refitting** project-specific or release-specific **models for an individual project**.

- A universal model would also **help interpret** basic **relationships between software metrics and defects**, potentially resolving inconsistencies among different studies

However…….

- To predict files with defects, **a suitable prediction model must be built for** a software project from **either itself** (within-project) or **other projects** (cross-project).
- One difficulty for building cross-project defect prediction models **is related to the variations in the distribution of predictors**
- Such variations exist among projects with different context factors (e.g., size and programming language).

To remedy this difficulty,

- The authors propose a **context-aware rank transformations for predictors** to address the variations in the distribution of predictors before fitting them to the universal defect prediction model.

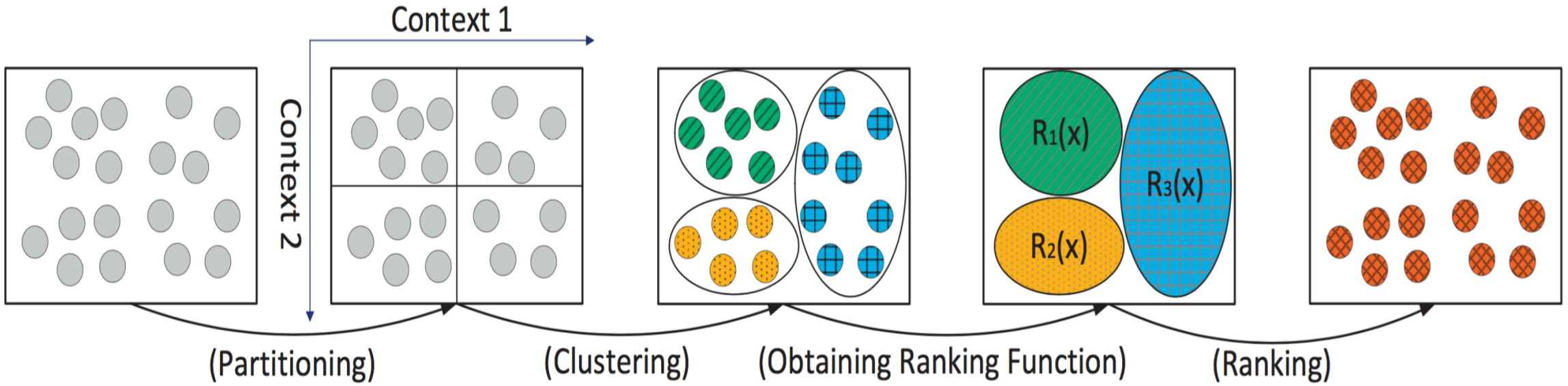
How is this done?

- The authors use **21 code metrics**, **5 process metrics**, and **6 context factors as predictors**.

- Concretely, the **context-aware approach** stratifies the **entire set of projects by context factors**, and **clusters the projects** with similar distribution of predictors.

- Rank transformations are then **derived using quantiles** of predictors for a cluster.
- After transformation, the predictors from different projects have exactly the same scales. **The universal model is then built based on the transformed predictors.**
- The authors applied their approach on 1,398 open source projects hosted on SourceForge and GoogleCode.

A Graphical View of the step-by-step approach



1) Partition the entire set of projects to non-overlapped groups based on the six aforementioned context factors;

project

clustered project

$R_1(x)$ $R_2(x)$ $R_3(x)$ ranking function

rank transformed project

2) Cluster the project groups with the similar distribution of predictor values;

3) Derive a ranking function for each cluster using every 10th quantiles of predictor values, in order to address the large variations in the distribution of predictors;

4) Apply the ranking functions to convert the raw values of predictors to one of the ten levels.

Technical Subtleties in building the Universal Defection Model

1. Context Factors,
2. Partition Projects,
3. Cluster Similar Projects,
4. Obtain Ranking Functions, and
5. Build the Model
6. Measure the Performance

Context Factors

- This study chose six context factors based on their availability to open source projects and previous work:

1. **Programming language (PL):** Due to the limitation of the metric computing tool, they only considered projects mainly written in C, C++, Java, C#, or Pascal.
2. **Issue Tracking (IT):** describes whether a project uses an issue tracking system or not.

3. **Total Lines of Code (TLOC):** describes the project size in terms of source code.
4. **Total Number of Files (TNF):** describes the project size in terms of files.
5. **Total Number of Commits (TNC):** describes the project size in terms of commits.
6. **Total Number of Developers (TND):** describes the project size in terms of developers.

Partition Projects

- It was assumed that **projects** with the **same context factors** have **similar distribution** of **software metrics**, and projects with different contexts might have different distribution of software metrics.
- Hence, the entire set of projects were stratified based on the six context factors.

Cluster Similar Projects

- To derive more accurate quantiles of a particular metric, **projects are grouped** that have **similar distribution of the metric**.
- Two distributions are considered similar if **their difference** is neither **statistically significant** nor **significantly large**.

Obtain Ranking Functions

- The **ranking function transforms the raw metric values** to relatively **predefined values** (i.e., ranging from one to ten). The transformed metrics have **exactly the same scales across projects**.
- Quantiles of metric values were used to formulate ranking functions. This is inspired by metric-based benchmarks, which often use the quantiles to derive thresholds of metrics to distinguish files of different quality related to defects.

Build the Model

- **Choice of modelling techniques:** There is no significant difference among different modelling techniques in the performance of defect prediction models.
- However, a past research finds that Bayes learners (i.e., Bayes Net and Naive Bayes) perform better when defect data contains noises, even up to 20%-35% of false positive and false negative noises in defect data.
- Based on this finding, they applied Naive Bayes as the modelling technique in the experiments.

Steps to build the universal defect prediction model:

- First, transform the raw values of each metric
- Before transforming a metric m_i for project p_j , identify context factors of project p_j and formulate a vector like

  < m_i, C++, useIT, moreTLOC, lessTNF, lessTNC, lessTND >.
- In order to locate the ranking functions, compare the vector of project p_j to the vectors of all clusters to determine which cluster project p_j belongs to.
- Apply the ranking functions of the identified cluster to transform the raw metric values of each file in project p_j to one of the ten levels. As a result, the transformed metrics have the scales ranging from one to ten.
- A universal defect prediction model is then built upon the entire set of projects using Weka tool.

Measure the Performance

- To evaluate the performance of the prediction models, a confusion matrix is computed, and used to calculate the precision, recall, false positive rate, F-measure, and g-measure, as well as the AUC

Actual \ Predicted	defective	non-defective
defective	true positive (TP)	false negative (FN)
non-defective	false positive (FP)	true negative (TN)

RQ1: Can a context-aware rank transformation provide predictive power comparable to the power of log transformation?

- They used a Wilcoxon rank sum test to compare the six performance measures.

Measures	LogTran	RankTran	p-value	Cohen's d
prec	**0.48**	**0.48**	0.71	-0.01
pd	0.57	**0.58**	0.31	0.03
fpr	0.36	**0.35**	0.43	0.04
F-measure	0.49	**0.50**	0.33	-0.03
g-measure	0.53	**0.54**	$4.7e$-03	-0.07*
AUC	0.61	**0.62**	0.14	-0.03

- The results show that the difference between the two transformations is small (i.e., less than 0.10). Hence, the rank transformation achieves comparable performance to log transformation.

RQ2: What is the performance of the universal defect prediction model?

- RQ2.1: Can context factors improve the predictive power?

The performance measures for the universal models built using code metrics (CM), code + process metrics (CPM), and code + process + contexts (CPMC), respectively.

Measures	CM	CPM	CPMC
prec	0.36	0.38	**0.40**
pd	**0.91**	0.83	0.86
fpr	0.87	0.76	**0.70**
F-measure	0.51	0.51	**0.55**
g-measure	0.23	0.36	**0.42**
AUC	0.58	0.60	**0.65**

RQ2: What is the performance of the universal defect prediction model?

- RQ2.2: Is the performance of the universal defect prediction model comparable to within-project models?

The results for Wilcoxon rank sum tests and mean values of six performance measures of within-project models (WM) and the universal models (UM). (* denotes statistical significance.)

Measures	WM	UM	p-value	Cohen's d
prec	**0.48**	0.45	2.34e-03	0.10*
pd	0.58	**0.63**	2.81e-11	-0.25*
fpr	**0.35**	0.45	6.76e-42	-0.48*
F-measure	**0.50**	0.46	2.19e-05	0.15*
g-measure	**0.54**	0.52	7.58e-14	0.14*
AUC	0.62	**0.64**	3.33e-04	-0.16*

- The results show that the universal model yields better recall and higher AUC than within-project models, possibly due to the fact that the defects in the files of similar properties are fixed in one project but overlooked in another.

RQ3: What is the performance of the universal defect prediction model on external projects?

- Generalizability of the universal model was examined by applying it on five external projects that are not hosted on SourceForge or GoogleCode (i.e., one Apache project: Lucene, and four Eclipse projects: Eclipse, Equinox, Mylyn, and PDE).

- The results show that the universal model provides a similar performance (in terms of AUC) as within- project models for the five projects.

Discussion

- How can this work be applied in software development industries?
- What are some of the threats to the validity of this work you can think of?
- Are there other vital context factors that may affect project clustering?
- Can we obtain better prediction accuracy with another classifier such as neural networks or support vector machines?