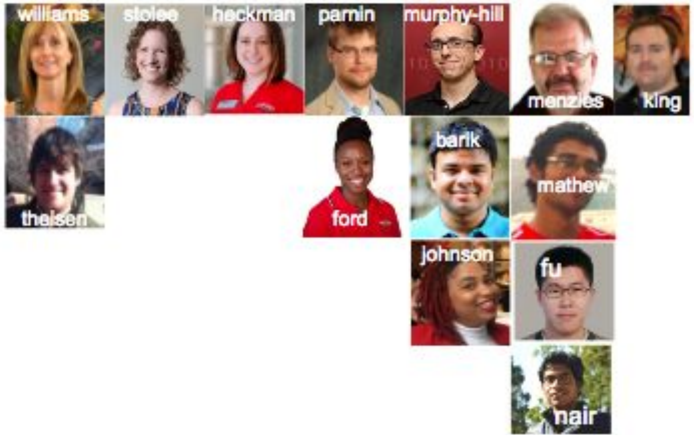
NC STATE UNIVERSITY

software

... if ~~engineering~~ then NC State ...



Revisiting Unsupervised Learning for Defect Prediction

Wei Fu **Tim Menzies**

wfu@ncsu.edu tim.menzies@gmail.com

http://weifu.us http://menzies.us

Find these slides at: http://tiny.cc/unsup

Sep, 2017

weifu.us

Tim Menzies
(advisor)

On the market

(expected graduation:May,2018)

Research Areas:

- Machine learning
- SBSE
- Evolutionary algorithms
- Hyper-parameter tuning

Publications:

FSE: 2
ASE: 1
TSE: 1
IST: 1
Under Review: 2

Yang et al.

reported: unsupervised predictors outperformed supervised predictors for effort-aware Just-In-Time defect prediction.

Implied: dramatic simplification of a seemingly complex task.

We

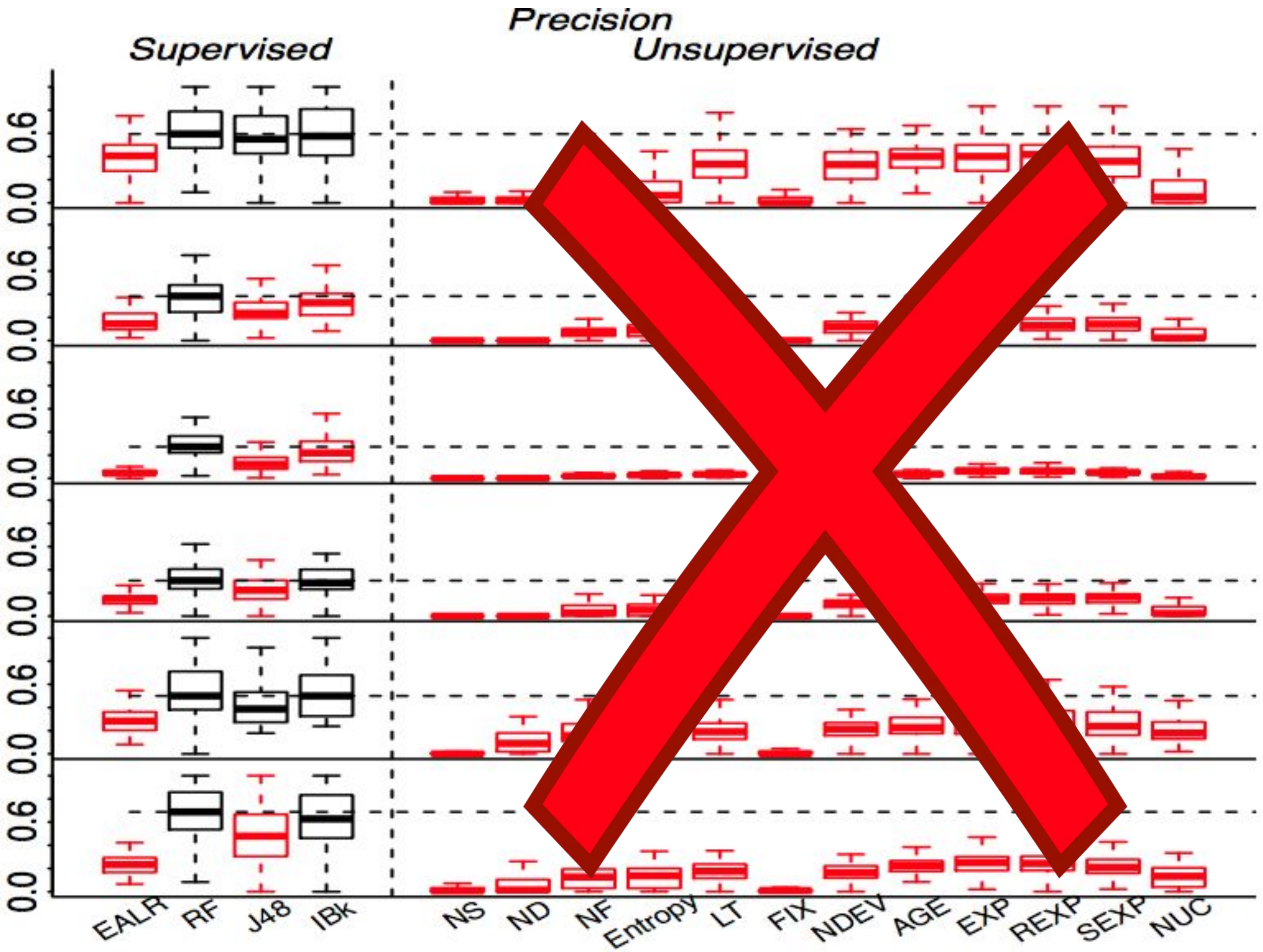
Repeated and **reflect** Yang et al. results.

General Lessons

Open Science

Yang's Unsup Methods in Precision

Red is bad!



Yang, Yibiao, et al. "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models." FSE'16

arXiv.org Effect

arXiv.org

Nanjing	Raleigh	Singapore Hangzhou	Nanjing	Raleigh
Mar 11, 2016	Feb 28, 2017	Apr 6, 2017	Apr 7, 2017	Oct, 2017
[Yang et al. FSE'16]	[Fu et al. FSE'17]	[Huang et al. ICSME'17]	[Liu et al. ESEM'17]	[Fu et al.arXiv'17]
Dr. Zhou	**Dr. Menzies**	**Dr. Xia & Dr. Lo**	**Dr. Zhou**	**Dr. Menzies**

Categories > Engineering & Computer Science > **Software Systems**

	Publication	h5-index	h5-median
1.	International Conference on Software Engineering	68	91
2.	IEEE Transactions on Software Engineering	52	80
3.	Journal of Systems and Software	51	73
4.	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	50	67
5.	ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	46	68
6.	Information and Software Techr	45	68
7.	ACM SIGSOFT International Sy	43	64
8.	Mining Software Repositories	39	57
9.	IEEE Software	38	54
10.	ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)	37	55
11.	ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)	37	51
12.	Empirical Software Engineering	37	48
13.	International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	36	55
14.	arXiv Software Engineering (cs.SE)	36	50
15.	International Symposium on Software Testing and Analysis	31	50
16.	IEEE/ACM International Conference on Automated Software Engineering (ASE)	31	44
17.	Software & Systems Modeling	31	43
18.	IEEE International Conference on Software Maintenance	29	41
19.	IEEE International Conference on Software Testing, Verification and Validation (ICST)	29	41
20.	arXiv Programming Languages (cs.PL)	29	40
X.	ACM Transactions on Software Engineering and Methodology(TOSEM)	?	?

Google Scholar

We Promote Open Science

A Large-Scale Empirical Study of Just-in-Time Quality Assurance

Yasutaka Kamei, *Member, IEEE*, Emad Shihab, Bram Adams, *Member, IEEE*, Ahmed E. Hassan, *Member, IEEE*, Audris Mockus, *Member, IEEE*, Anand Sinha, and Naoyasu Ubayashi, *Member, IEEE*

Abstract—Defect prediction models are a well-known technique for identifying defect-prone files or packages such that practitioners can allocate their quality assurance efforts (e.g., testing and code reviews). However, once the critical files or packages have been identified, developers still need to spend considerable time drilling down to the functions or even code snippets that should be reviewed or tested. This makes the approach too time consuming and impractical for large software systems. Instead, we consider defect prediction models that focus on identifying defect-prone ("risky") software *changes* instead of files or packages. We refer to this type of quality assurance activity as "Just-In-Time Quality Assurance," because developers can review and test these risky changes while they are still fresh in their minds (i.e., at check-in time) ... characteristics of a software change, such as the ... source and five commercial projects from multipl... defect with an average accuracy of 68 percent an... review changes, we find that using only 20 perce... the m... risky change... and thus red... the...

Index Terms—Maintenance, software metrics, ...

TSE'13

Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better Than Supervised Models

Yibiao Yang[1], Yuming Zhou[1*], Jinping Liu[1], Yangyang Zhao[1], Hongmin Lu[1], Lei Xu[1], Baowen Xu[1], and Hareton Leung[2]

[1] Department of Computer Science and Technology, Nanjing University, China

[2] Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

ABSTRACT

Unsupervised models do not require the defect data to build the prediction models and hence incur a low building cost and gain a wide application range. Consequently, it would be more desirable for practitioners to apply unsupervised models in effort-aware just-in-time (JIT) defect prediction if they can predict defect-inducing changes well. However, little is currently known on their prediction effectiveness in this context. We aim to investigate the predictive power of simple unsupervised models in effort-aware JIT defect prediction, especially compared with the state-of-the-art supervised models in the recent literature. We first use the most commonly used change metrics to build simple unsupervised models. Then, we compare these unsupervised models with the state-of-the-art supervised models under cross-validation, time-wise-cross-validation, and across-project prediction settings to determine whether they are of practical value. The experimental results, from open source software systems, show that many simple unsupervised models perform better than the state-of-the-art supervised models in effort-aware JIT defect prediction.

consecutive commits in a given period of time) that introduce one or several defects into the source code in a software system [37]. Compared with traditional defect prediction at module (e.g. package, file, or class) level, JIT defect prediction is a fine granularity defect prediction. As stated by Kamei et al. [13], it allows developers to inspect an order of magnitude smaller number of SLOC (source lines of code) to find latent defects. This could provide large savings in effort over traditional coarser granularity defect predictions. In particular, JIT defect prediction can be performed at check-in time [13]. This allows developers to inspect the code changes for finding the latent defects when the change details are still fresh in their minds. As a result, it is possible to find the latent defects faster. Furthermore, compared with conventional non-effort-aware defect prediction, effort-aware JIT defect prediction takes into account the effort required to inspect the modified code for a change [13]. Consequently, effort-aware JIT defect prediction would be more practical for practitioners, as it enables them to find more latent defects per unit code inspection effort. Currently, there is a significant strand of interest in developing effective effort-aware JIT defect prediction models [7, 13].

FSE'16

THANKS!

Our code & data at:

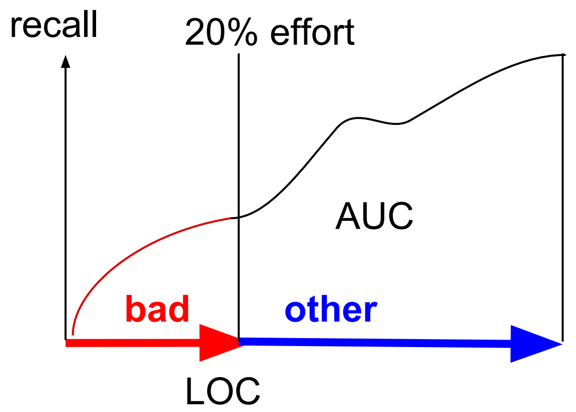
https://github.com/WeiFoo/RevisitUnsupervised

Methods

Yang et al's Method: Core Idea

Koru et al [Koru 2010] suggest that

"Smaller modules are proportionally more defect-prone and hence should be inspected first!"



Area under the curve of Recall/LOC

- **Bad** = modules predicted defective
- **Other** = all other modules
- Sort each increasing by LOC
- Track the recall

[Koru 2010]Koru, Gunes, et al. "Testing the theory of relative defect proneness for closed-source software." *Empirical Software Engineering* 15.6 (2010): 577-598.

Yang et al's Unsupervised Method

Build 12 unsupervised models, on **testing data**:

10% effort

20% effort

NF	NS	LT	FIX	ND	NDEV	EXP	REXP	SEXP	NUC	AGE	Entropy	LOC	Label
0	3	11	1	1	23	2	12	4	2	8	0.3	32	?
4	3	24	0	5	2	3	13	3	1	6	0.4	42	?
9	1	89	0	3	5	5	3	2	3	4	0.6	18	?
1	3	34	0	3	6	7	9	3	5	3	0.2	103	?
0	0	537	0	2	8	2	22	9	7	12	0.3	20	?
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Predicted as "Defective"

Yang et al report:

Aggregating performance over all projects, many simple unsupervised models perform better than supervised models.

OneWay

OneWay is not "the Way"

OneWay:

"The alternative way, maybe not the best way!"

--Wei

OneWay

Supervised data

X_1	X_2	LT	...	X_n	Label
0	3	11	...	3	0
4	3				
9	1				
1	3				
0	0				
⋮	⋮				

X_1	X_2	NS	...	X_n	Label
0	3	2	...	3	0
4	3				
9	1				
1	3				
0	0				
⋮	⋮				

X_1	X_2	NF	...	X_n	Label
0	3	2	...	3	1
4	3	12	...	2	1
9	1	23	...	8	0
1	3	56	...	1	0
0	0	90	...	2	0
⋮	⋮	⋮	⋱	⋮	⋮

Testing data

X_1	X_2	NS	...	X_n	Label
1	23	1	...	33	?
2	11	2	...	22	?
2	6	5	...	18	?
10	9	10	...	7	?
0	0	12	...	22	?
⋮	⋮	⋮	⋱	⋮	⋮

Build 12 learners

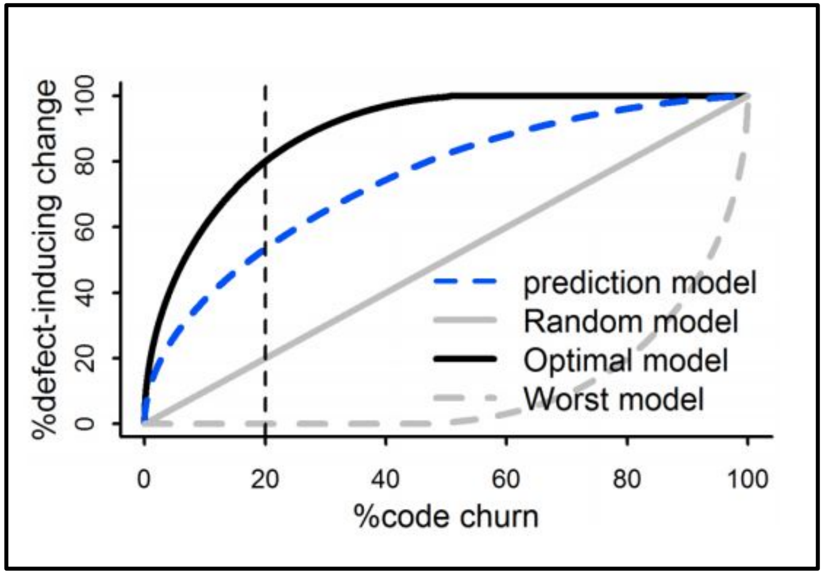
Select the best one(e.g.: NS)

Test with NS

Results

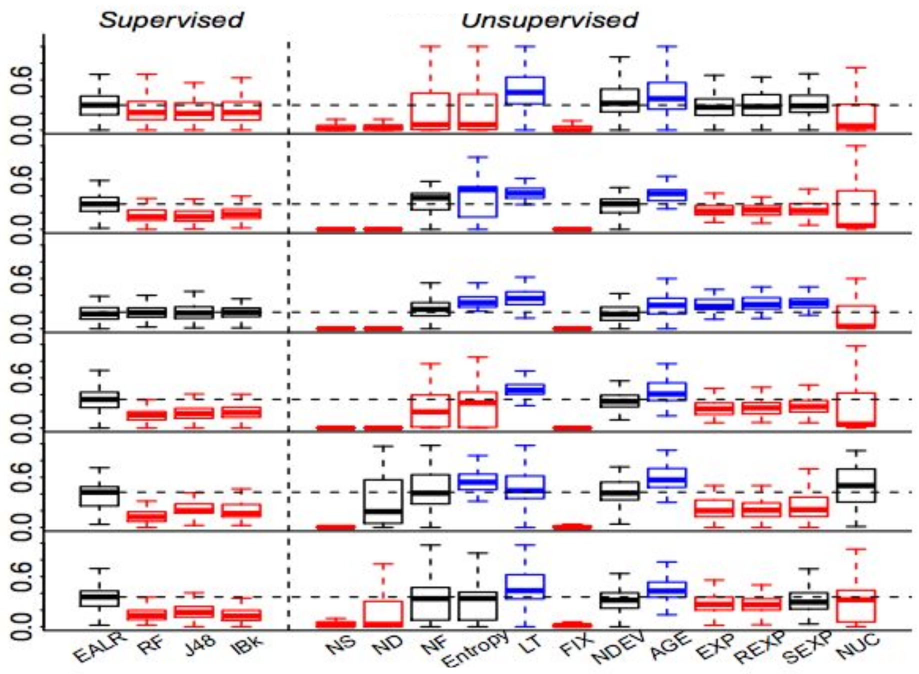
Our Evaluation Metrics

- Recall
- Popt
- **F1**
- **Precision**



$$P_{opt}(m) = 1 - \frac{S(optimal) - S(m)}{S(optimal) - S(worst)}$$

Our Result Format



Bugzilla
Platform
Mozilla
JDT
Columba
PostgreSQL

Blue: Better

Black: Similar

Red: Worse

Recall

Report results on a project-by-project basis.

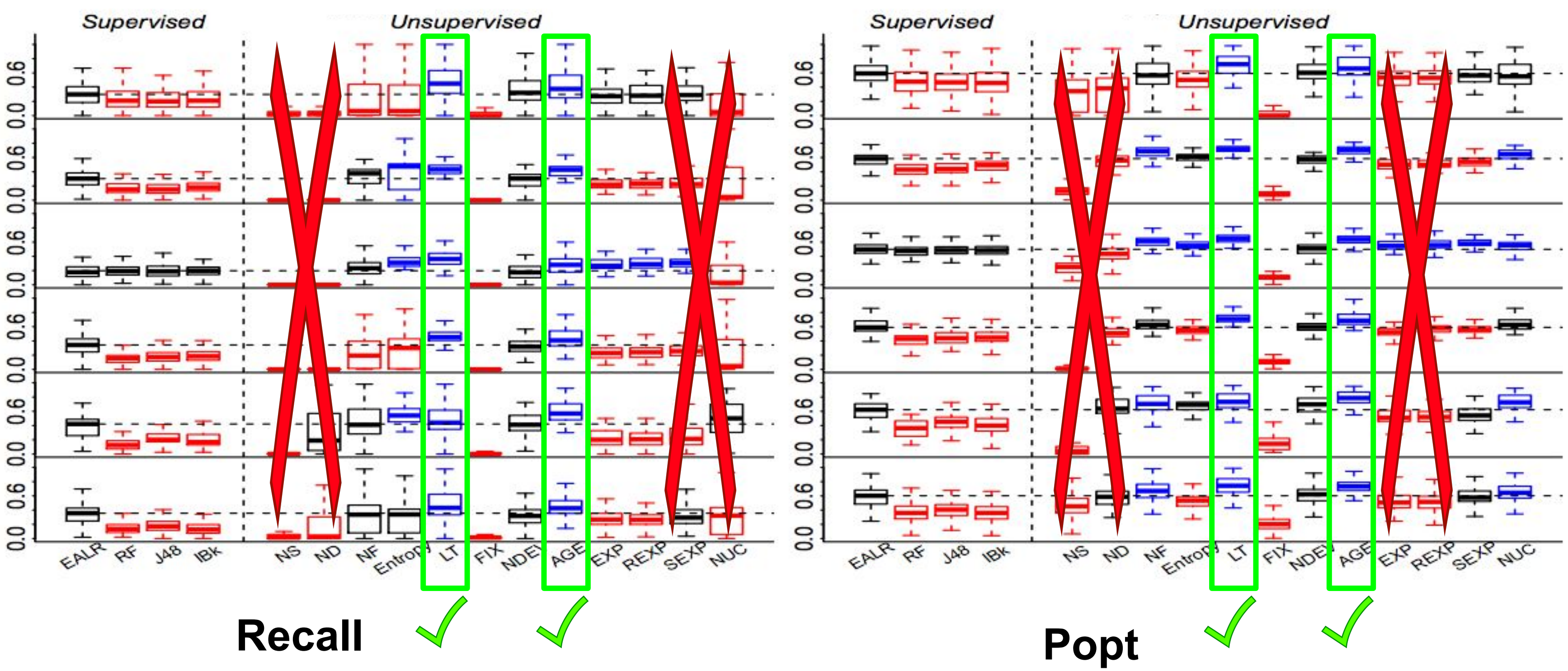
Research Questions

- All unsupervised predictors better than supervised?
- Is it beneficial to use supervised data?
- *OneWay* better than standard supervised predictors?

Research Questions

- All unsupervised predictors better than supervised?
- Is it beneficial to use supervised data?
- *OneWay* better than standard supervised predictors?

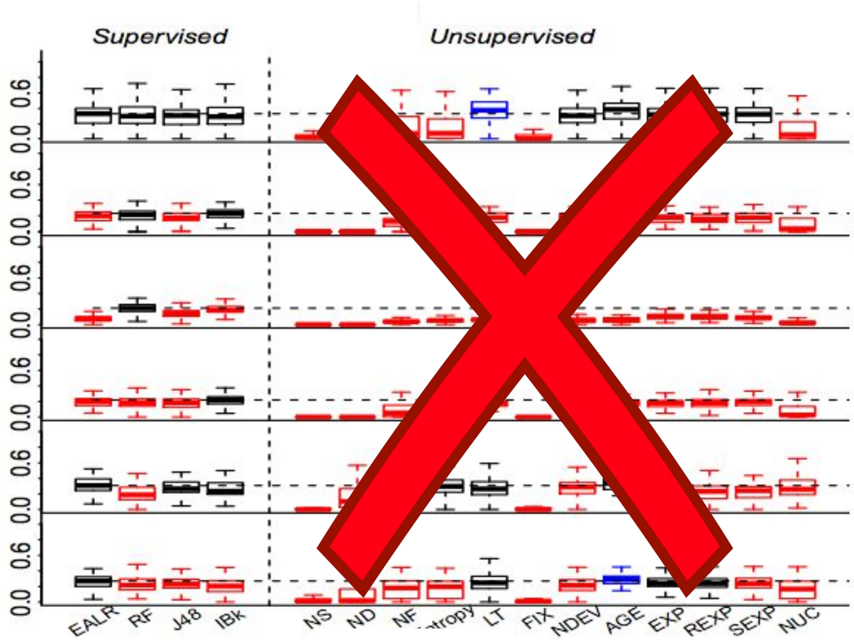
RQ1: All unsupervised predictors better?



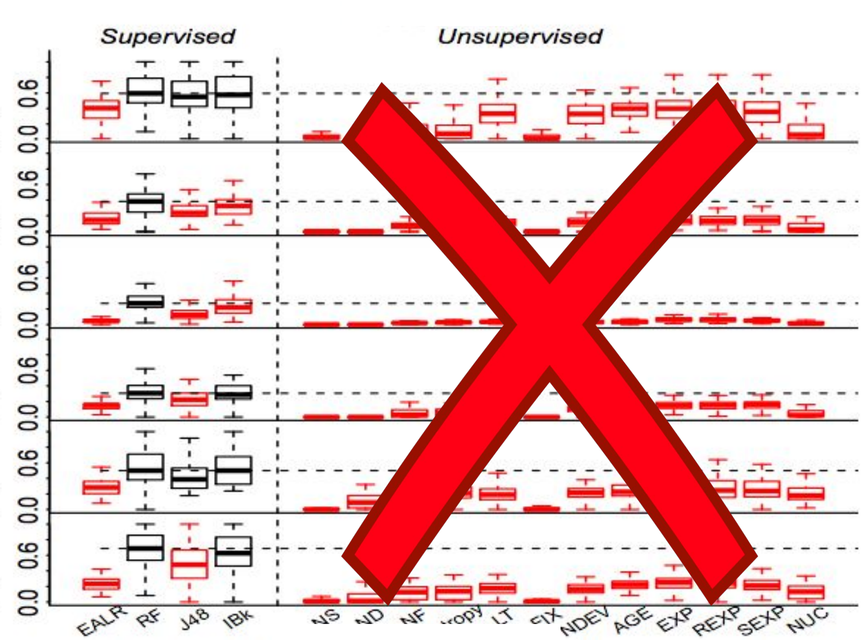
Recall: LT and AGE are better; Others are not.

Popt: the similar pattern as Recall.

RQ1: All unsupervised predictors better?



F1

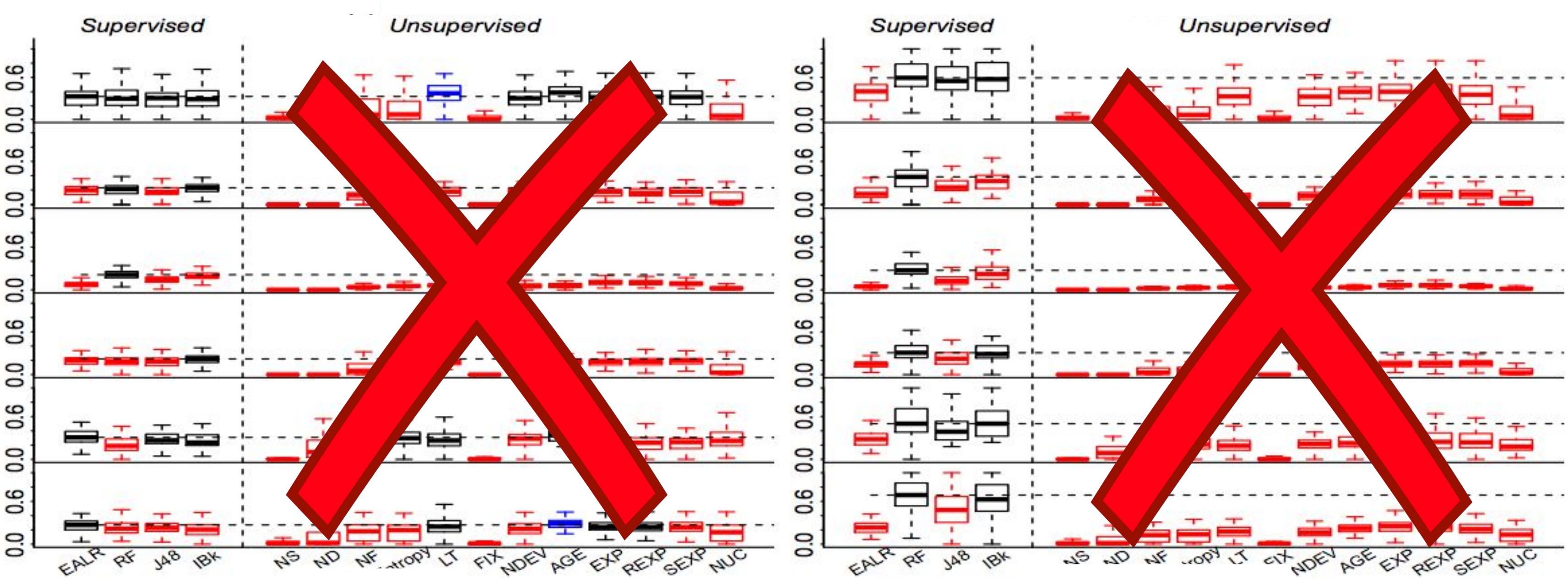


Precision

F1: Only two cases better, LT on Bugzilla; AGE on PostgreSQL;

Precision: All are worse than the best supervised learner!

RQ1: All unsupervised predictors better?



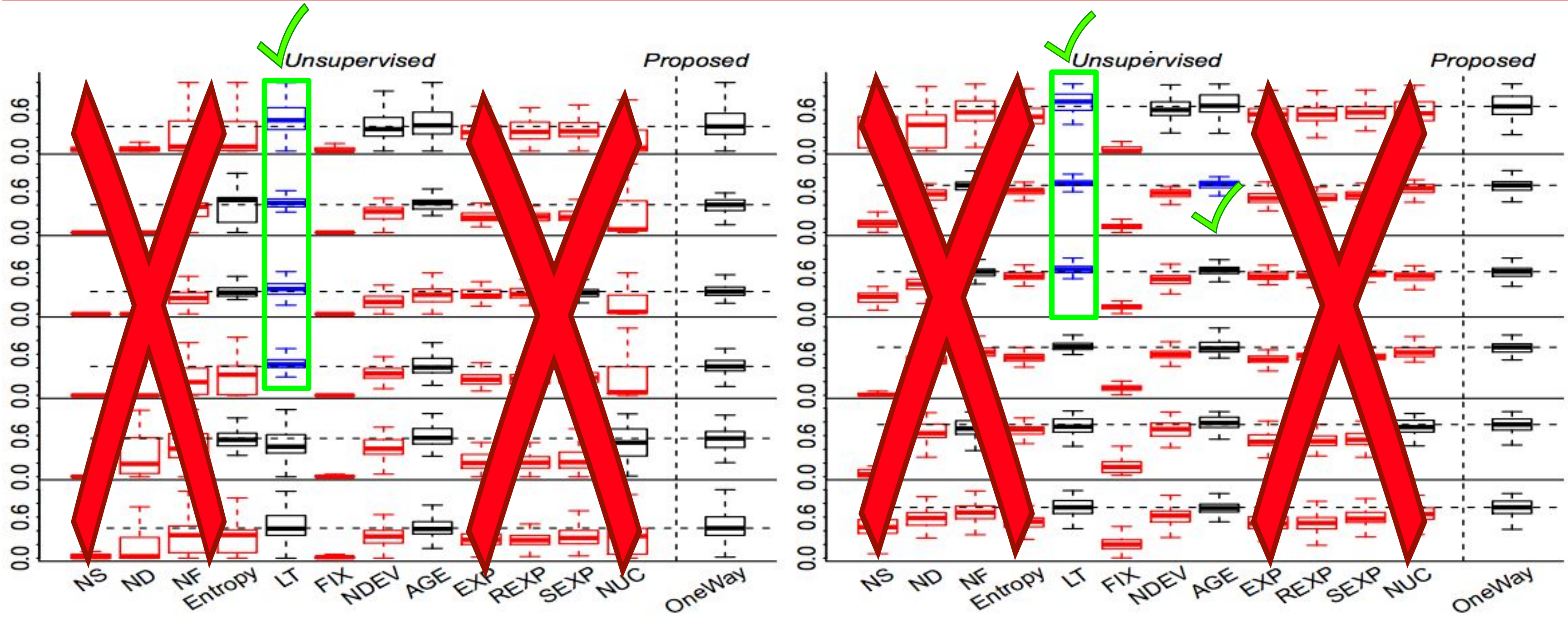
F1 **Precision**

Not all unsupervised predictors perform better than supervised predictors for each project and for different evaluation measures.

Research Questions

- All unsupervised predictors better than supervised?
- Is it beneficial to use supervised data?
- *OneWay* better than standard supervised predictors?

RQ2: Is it beneficial to use supervised data?

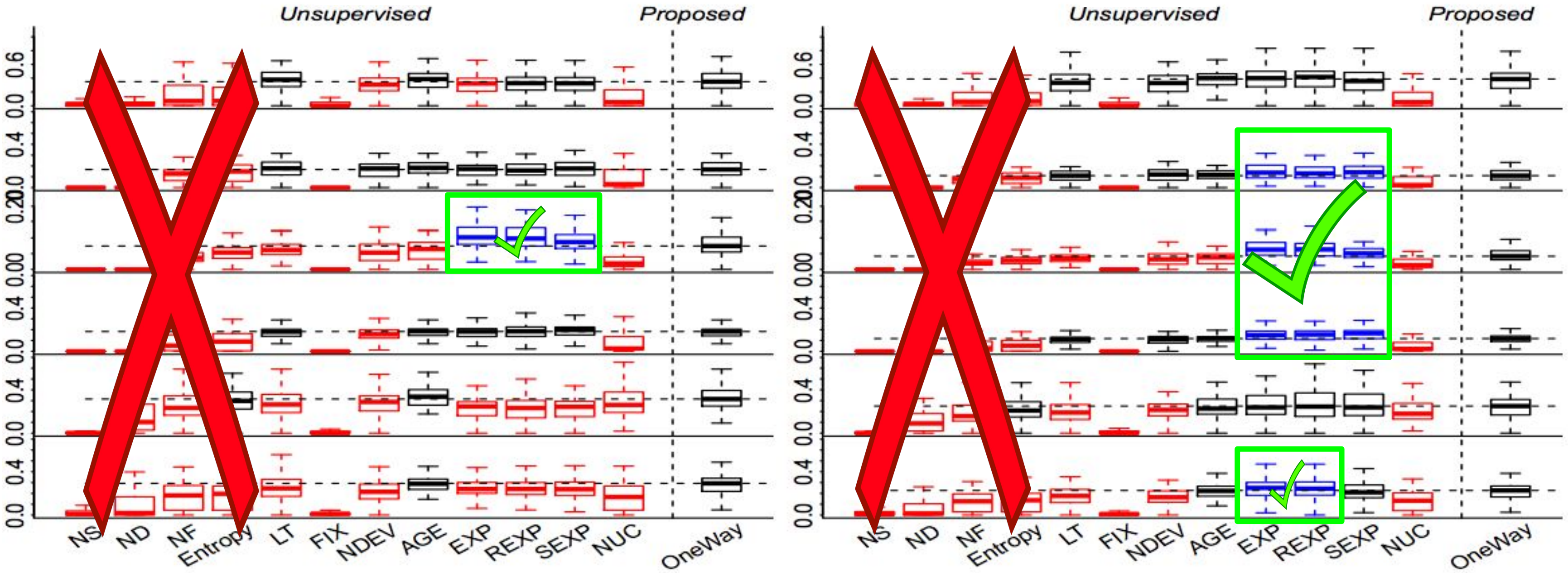


Recall **Popt**

Recall: *OneWay* was only outperformed by LT in 4/6 data sets.

Popt: The similar pattern as Recall.

RQ2: Is it beneficial to use supervised data?

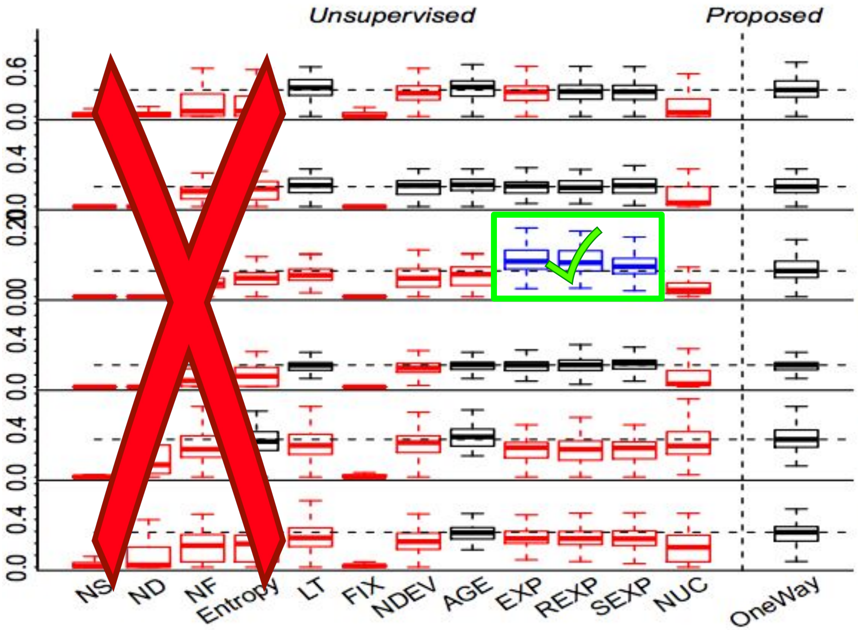


F1 Precision

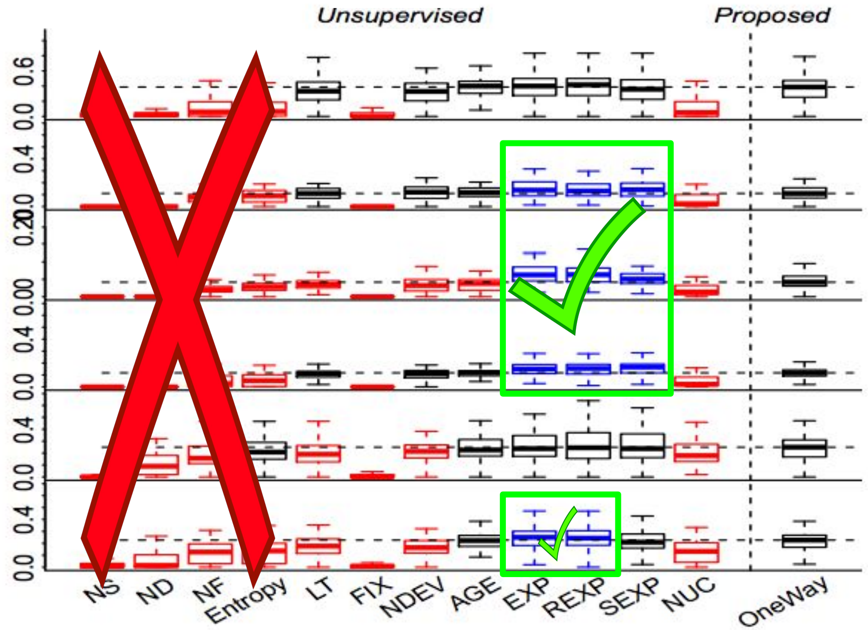
F1: EXP/REXP/SEXP performs better than *OneWay* only on Mozilla.

Precision: Similar as F1 but more data sets.

RQ2:Is it beneficial to use supervised data?



F1



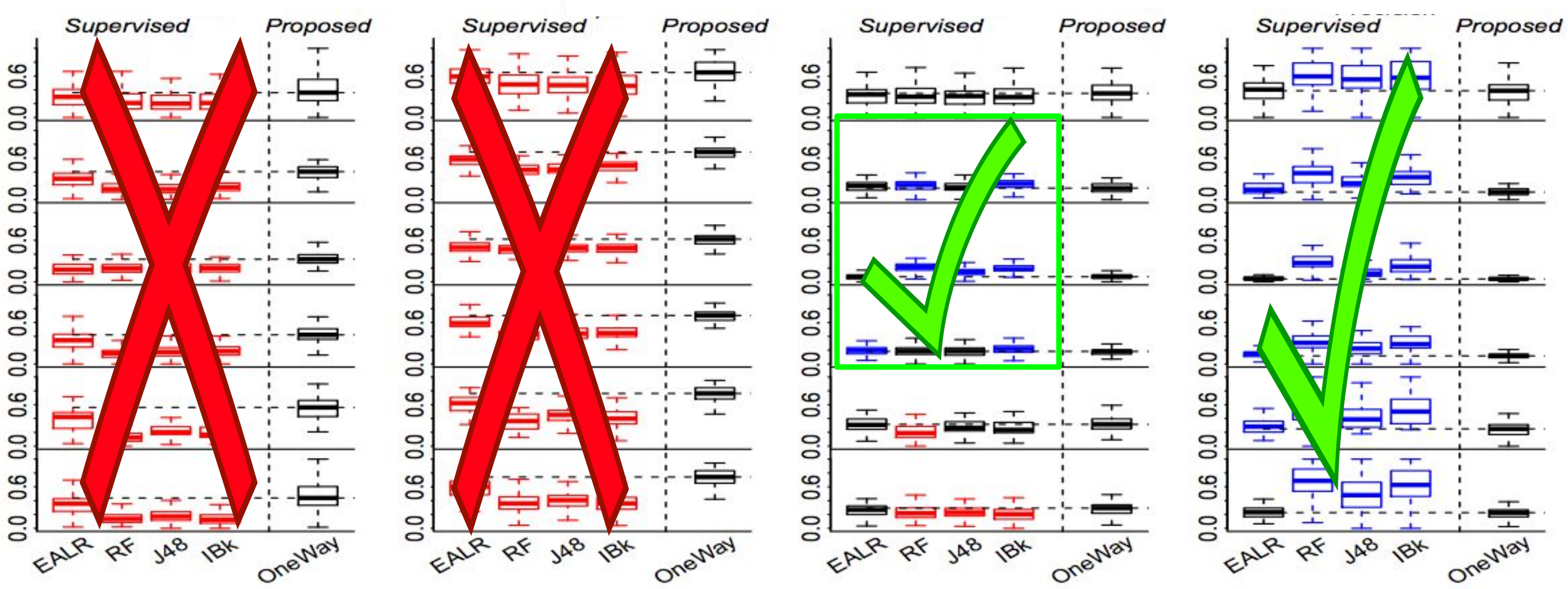
Precision

As a simple supervised predictor, OneWay performs better than most unsupervised predictors.

Research Questions

- All unsupervised predictors better than supervised?
- Is it beneficial to use supervised data?
- *OneWay* better than standard supervised predictors?

RQ3: *OneWay* better than supervised ones?



Recall **Popt** **F1** **Precision**

- *Better than supervised learners in terms of Recall & Popt;*
- *It performs just as well as other learners for F1.*
- *As for Precision, worse!*

Conclusion

Lessons Learned

- Don’t aggregate results over multiple data sets.
  - Different conclusions: Yang et al.

Lessons Learned

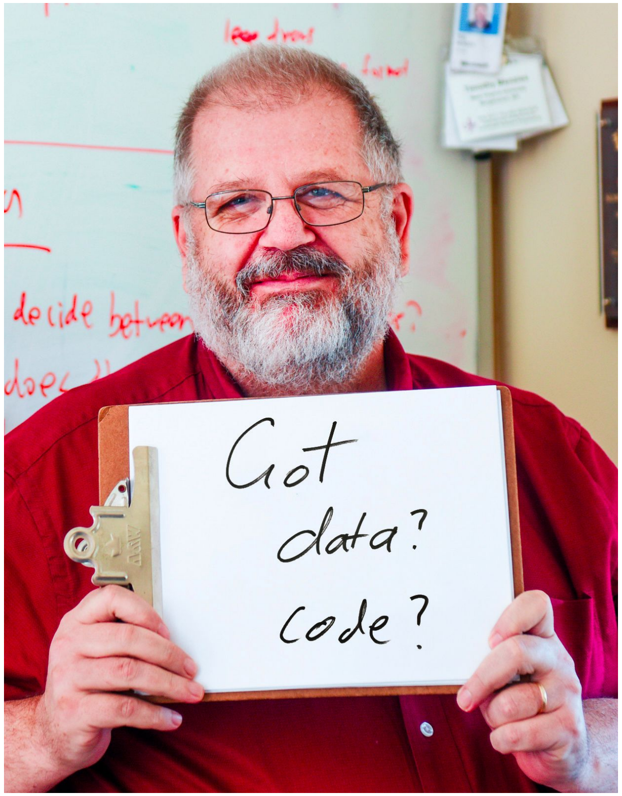
- Don’t aggregate results over multiple data sets.
  - Different conclusions: Yang et al.
- Do use supervised data.
  - Unsupervised learners’ are unstable.

Lessons Learned

- Don’t aggregate results over multiple data sets.
  - Different conclusions: Yang et al.
- Do use supervised data.
  - Unsupervised learners’ are unstable.
- Do use multiple evaluation criteria.
  - Results vary for different evaluation criteria.

Lessons Learned

- Don’t aggregate results over multiple data sets.
  - Different conclusions: Yang et al.
- Do use supervised data.
  - Unsupervised learners’ are unstable.
- Do use multiple evaluation criteria.
  - Results vary for different evaluation criteria.
- Do share code and data.
  - Easy reproduction.
- Do post pre-prints to arXiv.org.
  - Saved two years.



tiny.cc/seacraft

Why Seacraft?

- Successor of PROMISE repo, which contains a lot of SE artifacts.
- No data limits;
- Provides DOI for every submission (aka, easy citation);
- Automatic updates if linked to github project.

On the market

*(expected graduation: **May,2018**)*

weifu.us

(Machine learning, SBSE, Evolutionary algorithms, Hyper-parameter tuning)

OPEN science is FASTER science

Publications:

FSE: 2
ASE: 1
TSE: 1
IST: 1
Under Review: 2

Reference

1. Kamei, Yasutaka, et al. "Revisiting common bug prediction findings using effort-aware models." *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010.
2. Arisholm, Erik, Lionel C. Briand, and Eivind B. Johannessen. "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models." Journal of Systems and Software 83.1 (2010): 2-17.
3. Menzies, Tim, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors." *IEEE transactions on software engineering* 33.1 (2007): 2-13.
4. Rahman, Foyzur, Daryl Posnett, and Premkumar Devanbu. "Recalling the imprecision of cross-project defect prediction." *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012.
5. Mende, Thilo, and Rainer Koschke. "Effort-aware defect prediction models." *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. IEEE, 2010.
6. D'Ambros, Marco, Michele Lanza, and Romain Robbes. "Evaluating defect prediction approaches: a benchmark and an extensive comparison." *Empirical Software Engineering* 17.4-5 (2012): 531-577.
7. Kamei, Yasutaka, et al. "A large-scale empirical study of just-in-time quality assurance." *IEEE Transactions on Software Engineering* 39.6 (2013): 757-773.
8. Yang, Yibiao, et al. "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models." *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016.