Are Fix-Inducing Changes a Moving Target?

A Longitudinal Case Study of Just-In-Time Defect Prediction

Shane McIntosh
McGill University
Montréal, Canada
shane.mcintosh@mcgill.ca

Yasutaka Kamei
Kyushu University
Fukuoka, Japan
kamei@ait.kyushu-u.ac.jp

ABSTRACT

Change-level defect prediction [5], a.k.a., Just-In-Time (JIT) defect prediction [1], is an alternative to module-level defect prediction that offers several advantages. First, since code changes are often smaller than modules (e.g., classes), JIT predictions are made at a finer granularity, which localizes the inspection process. Second, while modules have a group of authors, changes have only one, which makes triaging JIT predictions easier. Finally, unlike module-level prediction, JIT models can scan changes as they are being produced, which means that problems can be investigated while design decisions are still fresh in the developers' minds.

Despite the advantages of JIT defect prediction, like all prediction models, they assume that the properties of past events (fix-inducing changes) are similar to the properties of future ones. This assumption may not hold—the properties of fix-inducing changes in one time period may be different from those of another period. In our paper [4], we set out to address the following central question:

Do the important properties of fix-inducing changes remain consistent as systems evolve?

To address our central question, we train JIT models using six families of code change properties, which are primarily derived from prior studies [1–3, 5]. These properties measure: (a) the magnitude of the change (*Size*); (b) the dispersion of the changes across modules (*Diffusion*); (c) the defect proneness of prior changes to the modified modules (*History*); (d) the experience of the author (*Auth. Exp.*) and (e) code reviewer(s) (*Rev. Exp.*); and (f) the amount of participation in the review of the code change (*Review*).

Through a longitudinal case study of 37,524 changes from the rapidly evolving Qt and OpenStack systems, we find that the answer to our central question is no:

- JIT models lose a large proportion of their discriminatory power (AUC) and calibration (Brier) scores one year after being trained.
- The magnitude of the importance scores of code change properties fluctuate as systems evolve (e.g., Figure 1 shows fluctuations across six-month periods of OpenStack).
- These fluctuations can lead to consistent overestimates (and underestimates) of the future impact of the studied families of code change properties.

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05.
https://doi.org/10.1145/3180155.3182514

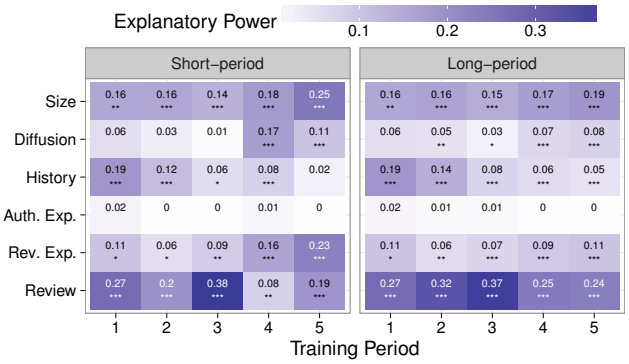

Figure 1: The importance of code change properties over time in OpenStack. Shade indicates magnitude while asterisks indicate that Wald χ^2 test p < * 0.05, ** 0.01, * < 0.001.**

To mitigate the impact on model performance, researchers and practitioners should add recently accumulated data to the training set and retrain JIT models to contain fresh data from within the last three months. To better calibrate quality improvement plans (which are based on interpretation of the importance scores of code change properties), researchers and practitioners should put a greater emphasis on larger caches of data, which contain at least six months worth of data, to smooth the effect of spikes and troughs in the importance of properties of fix-inducing changes.

ACM Reference Format:
Shane McIntosh and Yasutaka Kamei. 2018. Are Fix-Inducing Changes a Moving Target?. In *Proceedings of ICSE '18: 40th International Conference on Software Engineering , Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18),* 1 pages.
https://doi.org/10.1145/3180155.3182514

REFERENCES

[1] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Transactions on Software Engineering (TSE)* 39, 6 (2013), 757–773.
[2] Sunghun Kim, E. James Whitehead, Jr., and Yi Zhang. 2008. Classifying Software Changes: Clean or Buggy? *IEEE Transactions on Software Engineering (TSE)* 34, 2 (2008), 181–196.
[3] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. 2015. Investigating Code Review Quality: Do People and Participation Matter?. In *Proc. of the 31st Int'l Conf. on Software Maintenance and Evolution (ICSME).* 111–120.
[4] Shane McIntosh and Yasutaka Kamei. 2017. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. *IEEE Transactions on Software Engineering* (2017), To appear. https://doi.org/10.1109/TSE.2017.2693980
[5] Audris Mockus and David M. Weiss. 2000. Predicting Risk of Software Changes. *Bell Labs Technical Journal* 5, 2 (2000), 169–180.