Title: 16_MSR_MUBench A Benchmark for API-Misuse Detectors

Author: S Amann, S Nadi, HA Nguyen, TN Nguyen, M Mezini

Background:
No work empirically analyzes the prevalence of API misuses compared to other types of bugs or shows which kinds of misuses a particular technique detects. This makes it hard to judge the impact of the detectors in general, to assess their capabilities and to compare them with one another.

Conclusion:

1. API misuse is rare but almost always cause crashes. Since they find that many API misuses lead to obviously spurious behavior, they hypothesize that many are already ruled out during development through testing. Thus, developers might face many misuses that they can not find by reviewing bugs datasets.
2. A benchmark of API misuses with 89 API-misuses and from 33 real-projects.

Method:

Useful Information:

1. **API misuses:** violations of usage patterns and API contracts
2. **Bug Bechmarks:**
    1. BugClassify - K. Herzig, S. Just, and A. Zeller. It's Not a Bug, It's a Feature: How Misclassication Impacts Bug Prediction. ICSE'13, pages 392{401. IEEE Press, 2013.
    2. Defects4J - R. Just, D. Jalali, and M. D. Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. ISSTA'14, pages 437{440. ACM, 2014.
    3. IBUGS - V. Dallmeier and T. Zimmermann. Extraction of Bug Localization Benchmarks from History. ASE'07, pages 433{436. ACM, 2007.
    4. QACRASHFIX - Q. Gao, H. Zhang, J. Wang, Y. Xiong, L. Zhang, and H. Mei. Fixing Recurring Crash Bugs via Analyzing Q&A Sites. ASE'15, pages 307{318, 2015
3. **Using the Benchmark:** Evaluation of API-misuse detectors.
4. **Classification of Misuses by characteristics:**
    1. Superfluous call -
    2. Missing call -
    3. Wrong call -
    4. Missing Preconditions -
        1. predicate, calls
        2. null check on a reference, parameters
        3. value constraint
    5. Missing Catch -
    6. Missing Finally -
    7. Ignored Result - return value

Useful Related Work:

Benchmark for C

[1] C. Cifuentes, C. Hoermann, N. Keynes, L. Li, S. Long, E. Mealy, M. Mounteney, and B. Scholz. BegBunch: Benchmarking for C Bug Detection Tools. DEFECTS'09, pages 16{20. ACM, 2009.

Benchmark For JAVA

[2] V. Dallmeier and T. Zimmermann. Extraction of Bug Localization Benchmarks from History. ASE'07, pages 433{436. ACM, 2007

[4] Q. Gao, H. Zhang, J. Wang, Y. Xiong, L. Zhang, and H. Mei. Fixing Recurring Crash Bugs via Analyzing Q&A Sites. ASE'15, pages 307{318, 2015.

[6] K. Herzig, S. Just, and A. Zeller. It's Not a Bug, It's a Feature: How Misclassication Impacts Bug Prediction. ICSE'13, pages 392{401. IEEE Press, 2013.

[7] R. Just, D. Jalali, and M. D. Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. ISSTA'14, pages 437{440. ACM, 2014.

Referenced statements:

Over the last 15 years, researchers proposed a multitude of automated bug-detection approaches. [13] [13] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated API Property Inference Techniques. IEEE Trans. Soft. Eng., 39:613{637, 2013.

Thoughts:

1. Combine JML with this benchmark and check as a case study? whether JML can describe all the properties?
2. No detector that identifies superfluous calls