A Bug Finder Refined by a Large Set of Open-Source Projects

Jaechang Nam, Song Wang, Yuan Xib, Lin Tan

Handong Global University, Pohang, Gyeongsangbuk-do, Korea
University of Waterloo, Waterloo, ON, Canada

Review by Anurag Bhattacharjee

Static bug detection tools have been widely adopted in the industry. But the biggest obstacle behind static bug detection techniques being beneficial to developers are false positives, i.e., reported bugs that are not actually bugs or don't need developers' consent.

This research paper focuses on designing bug detection rules by manually examining a large number of software projects and iteratively refining them from the feedback until all false positives are filtered out. The outcome is a set of 10 bug detection rules and a bug finder, FEEFIN (<u>Fee</u>dback-based bug <u>Fin</u>der), based on those rules. Upon applying FEEFIN on 1880 real-world projects on Github it could actually achieve the goal of 0 FP (false positive) by grouping projects in three groups and continuously refining the rules from the feedback from every iteration. In total from the three groups, 57 true bugs were detected and 44 among them were actually fixed by the developers. This completely establishes the observation that feedback based rule design can mitigate false positives.

The industry has been adopting various static bug detection techniques but a large number of false alarms has always been the major challenge. Researchers have proposed techniques to filter out false positives by prioritizing all reported warnings and focusing on warnings with top priority, building statistical models to classify false positives and true warnings, combining static and dynamic analysis to filter out false positives. But 30-90% of reported warnings by static bug detection tools are still false positives. So, this research proposes an iterative manual method to design bug detection rules from bug-fixing patches and refine them from the feedback of every iteration by applying them in a large number of software projects.

The goal of this study is to explore the effectiveness of feedback-based bug detection rule design in mitigating the challenge of false positives by reducing them and also detecting new bugs. The process of achieving this goal and evaluating the result consists of three steps:

- **Feedback-based bug detection rule design**:
  An iterative manual process that repeatedly refines by examining patches from different software and the false positives detected from the feedback.
- **FEEFIN**:
  A static bug-finder based on the ten rules designed on the previous step. Among them, six are completely new and four of them produce considerably fewer false positives.

- **Case study and empirical evaluation of the rule design**:
  Considered in two scenarios, (i) Just-in-time(JIT) which detects bug before developer commit changes, (ii) Snapshot FEEFIN which detects bugs from the latest snapshot of the project.

Designing the feedback-based rule design and the implementation of FEEFIN is a three-step process:

(1) **Manual Patch Analysis**:
1,622 small patches whose number of modified lines are at most five, were analyzed from four open-source projects, Lucene, Jackrabbit, Hadoop-common, and HBase. An efficient way to find common bug introducing patches was used by tracking issues from version controlling system, i.e., Git other than using SZZ Algorithm which sometimes can give noisy outputs by reporting feature requests as bug reports.
Error-prone coding practices are one of the most common bug patterns. Analyzing all the simple and common mistakes from the small patches, initially, 10 bug patterns were detected. Six of them are completely new and the other four though overlaps with the existing rules, still provides significantly less false positives compared to the existing ones. But these initial rules also generated false positives which were refined in the next step.

(2) **Feedback-based rule design:**
A feedback-based rule design is an iterative manual process that improves the rules by using the false positives detected from the results by applying the rules on a large set of software projects. With a large set of software projects, it is much easier to garner the problems in the existing rules and further improve them. This is a benefit gained from mining software repositories. To gain feedback for the initial rules, these rules were applied to 599 popular projects on Github.
On the other hand, validating a rule on a large set of software projects helps a tool designer decide whether the rule for a bug patter should be included in the rule depending on their certainty because some patterns may look like a bug but they were intentionally created by the developers depending on their use cases.

(3) **FEEFIN:**
Bug finder FEEFIN is based on abstract syntax tree(AST) and heuristics that embody the questions formed from the feedback-based detection rule design. This was applied in two detection scenarios, i.e., JIT (Just-In-Time) and Snapshot FEEFIN.
Based on the past commits of the 599 projects, scenario was simulated to validate FEEFIN in JIT and 160 true positives were found with only one false positive which is only 0.625%. For detecting a bug as a true positive it was made sure it is truly a bug and not a cosmetic change that has been fixed by the developers.
But this preliminary result is from the same project where the feedback based rule design was conducted. So, to evaluate it on unknown scenarios, Snapshot FEEFIN was applied on 1880 JAVA projects on Github including the 599 projects. It could detect 129 potential bugs among them 97 cases were reported to issue tracking systems and 54 were confirmed by developers as true positives and only 9 as false positives.

The case study focuses on how feedback-based rule design can detect unknown bugs while reducing the number of false positives and also the benefits of a feedback-based design. A pattern was considered new when they were not present in the existing bug detection tools namely, PMD, FindBugs, Facebook Infer, and Google Error Prone. The number of false positives was reduced on every iteration and lead to 0 false positives on the 3rd group of implication. This implies the importance of iteratively verifying the rules on a large number of projects to overcome the challenge of the static bug finders. Though detecting rules by mining software is a manual process, it's worth it and it's a one-time event.

Some of the derived bug patterns may not be directly applied to software built on other programming languages. Also, FEEFIN worked on less severe bugs so the practical aspect of how FEEFIN can save a company's debugging effort is unknown.