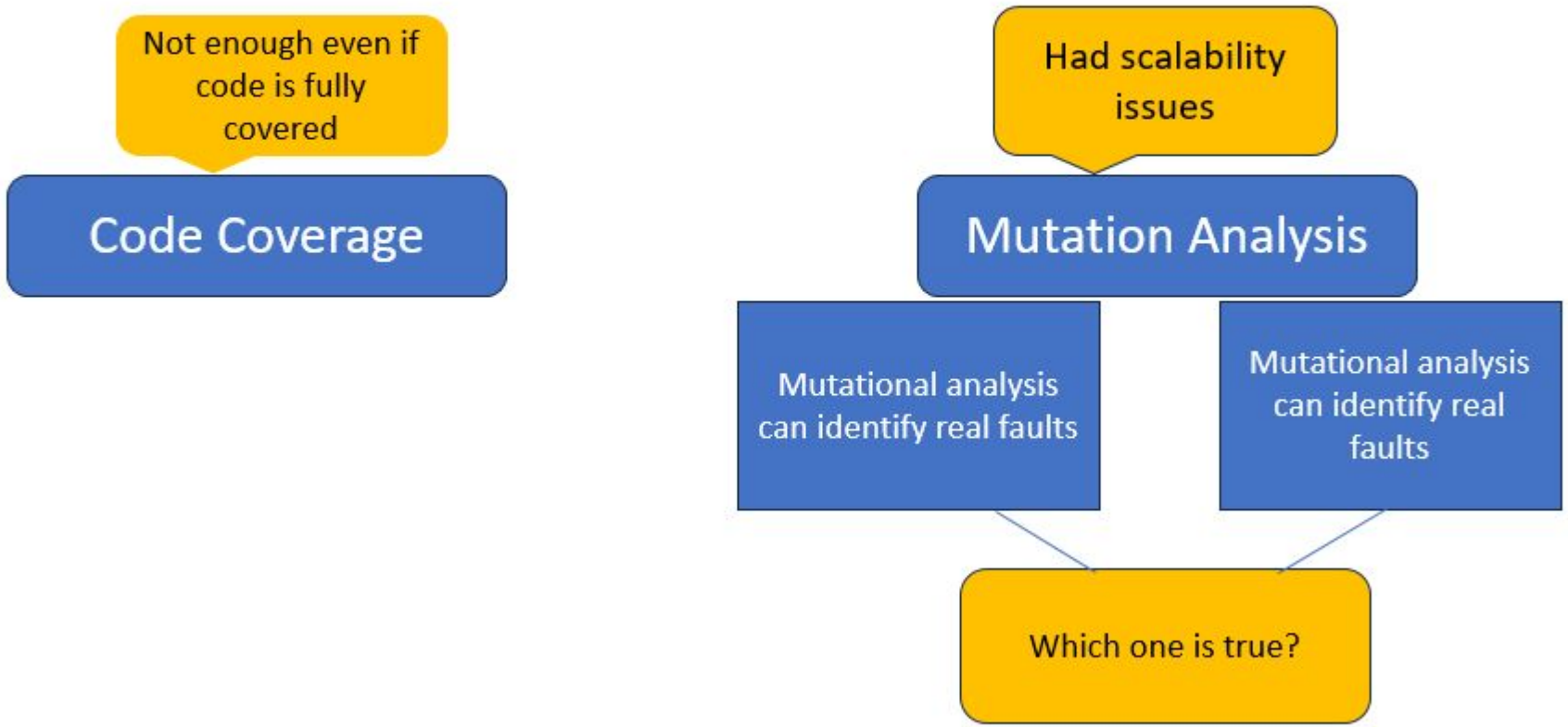
Does mutation testing improve testing practices ?

Goran Petrovic, Marko Ivankovic, Gordon Fraser
and Rene Just

ICSE 2021

Motivation and Background



Problem

With modern research being able to scale mutation testing, does it really help the developers in practice?

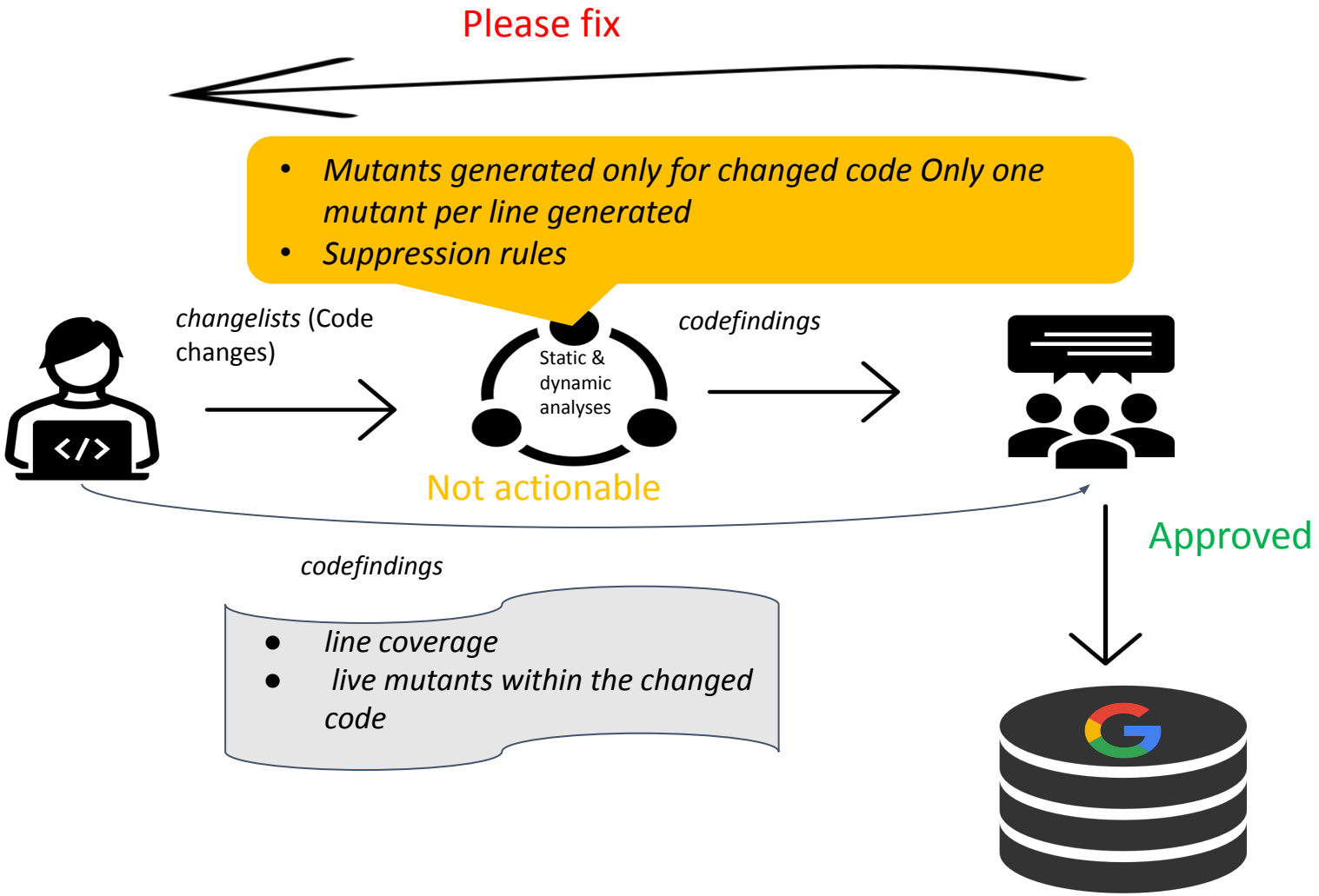
No research has previously evaluated whether mutants are meaningful and actionable test goals?

Dataset & Research Question

15 Million
Mutants

- **RQ1**: How does continuous mutation testing affect how much test code developers produce? (Effects on testing quantity)
- **RQ2**: How does continuous mutation testing affect the survivability of the mutants on a project ?(Effects on testing quality)
- **RQ3**: Are reported mutants coupled with real software faults ? Can tests written based on mutants improve test effectiveness of real software faults ?
- **RQ4**: Are mutants generated for a given line redundant ? Is it sufficient in practice to select a single mutant per line ?

Mutation testing at Google



Mutation operators

- *AOR*
- *LCR*
- *ROR*
- *UOI*
- *SBR*

AOR – Arithmetic operator replacement, LCR – Logical connector replacement, ROR – Relational operator replacement, UOI – Unary operator insertion, SBR – Statement block removal

Approach and Dataset: RQ1 & RQ2

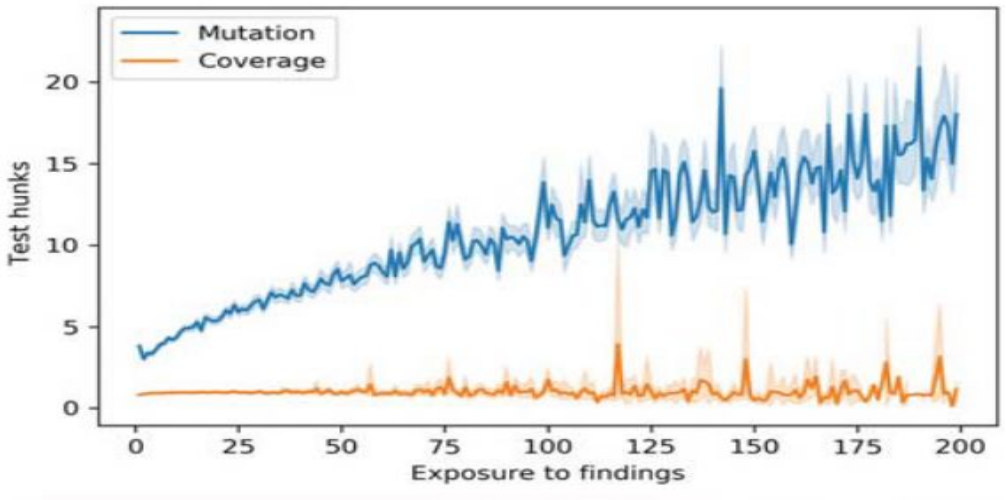
For each code change, identified edited test files that correspond to that change

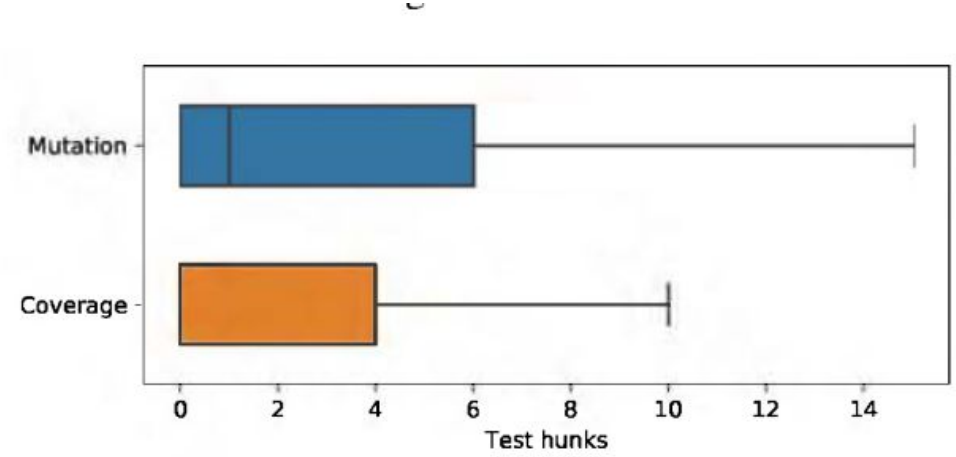
- Test Hunks
- Exposure
- Evaluate # of test-hunks for each file over the file's exposure
- Mutant Survivability

	Mutant Dataset	Coverage dataset (Baseline)
Number of mutants	14,730,562	Line coverage calculation enabled
Number of code changes	662584	8,788,791
Number of files	446604	3398085

Evaluation-RQ1(Testing Quantity)

As exposure to mutation testing increases, developers tend to write more tests





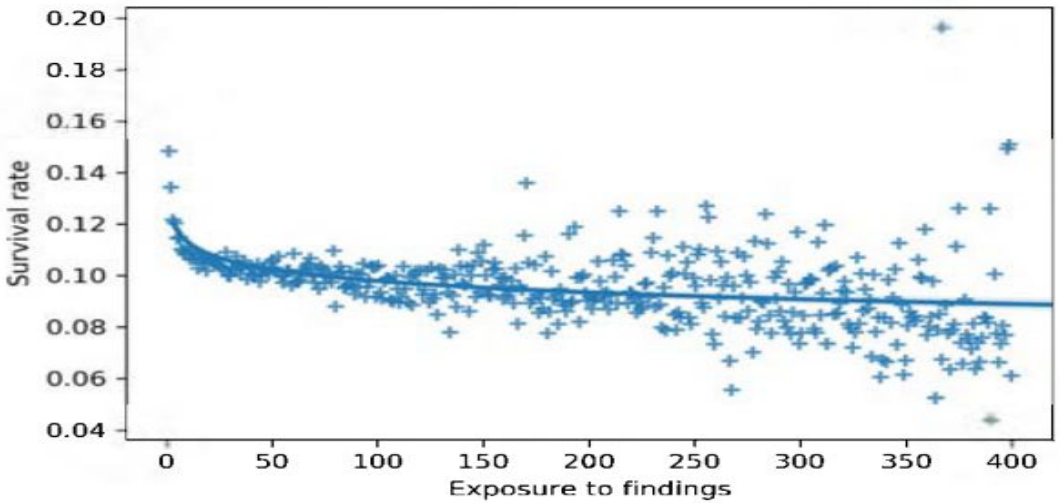
For Mutation dataset:

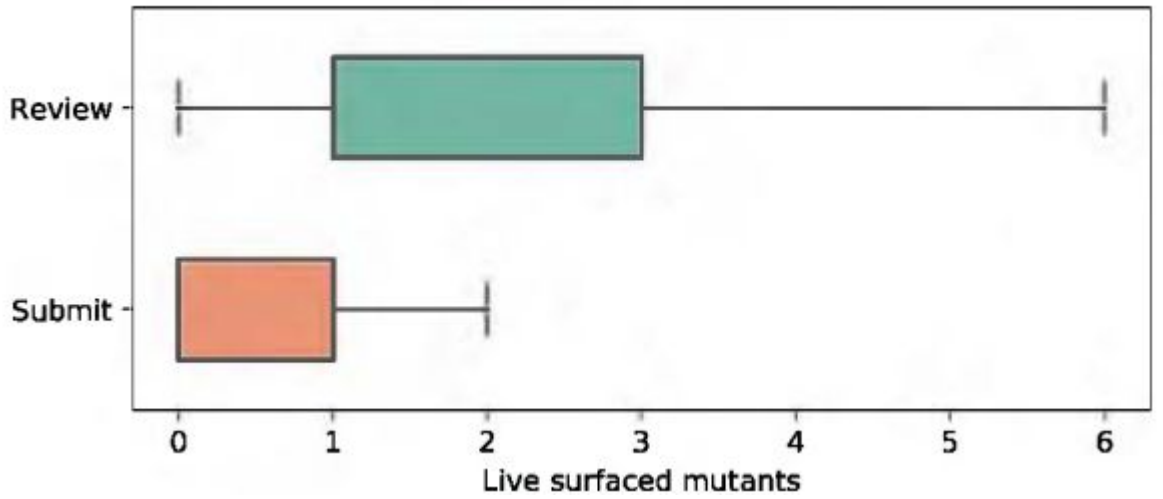
- The number of test hunks changed is larger
- The larger the exposure, more test hunks are changed on average
- Shows mutation testing has a positive long-term effect on testing quantity

- This suggests that mutants are actionable and guide developers in writing additional tests.

Evaluation-RQ2(Testing Quality)

As exposure to mutation testing increases, developers tend to write stronger tests in response to mutants





- The more mutant a file sees, more likely that the existing tests kill those mutants

- If additional tests for a changed file improve overall test suite quality, then future changes to the same file should see fewer surviving mutants.

Approach & Dataset: RQ3 & RQ4

Dataset

1) Found explicit high priority bug-fixing changes in the version control history

2) Analyzed changes and retain suitable bugs

	Bugs		Test targets		Affected files		Affected lines	
	Count	Ratio	Median	Mean	Median	Mean	Median	Mean
C++	736	49.0%	12	35.0	3	4.8	52	174.7
Java	501	33.4%	22	42.5	3	4.9	50	263.3
Python	180	12.0%	10	30.7	2	3.5	24	83.6
Go	85	5.7%	5	22.2	3	3.6	41	516.9
Total	1502	100.0%	13	36.2	3	4.6	46.5	212.7

Approach

Coupling effect : A test suite that detects simple faults is sensitive enough to likely detect complex faults as well

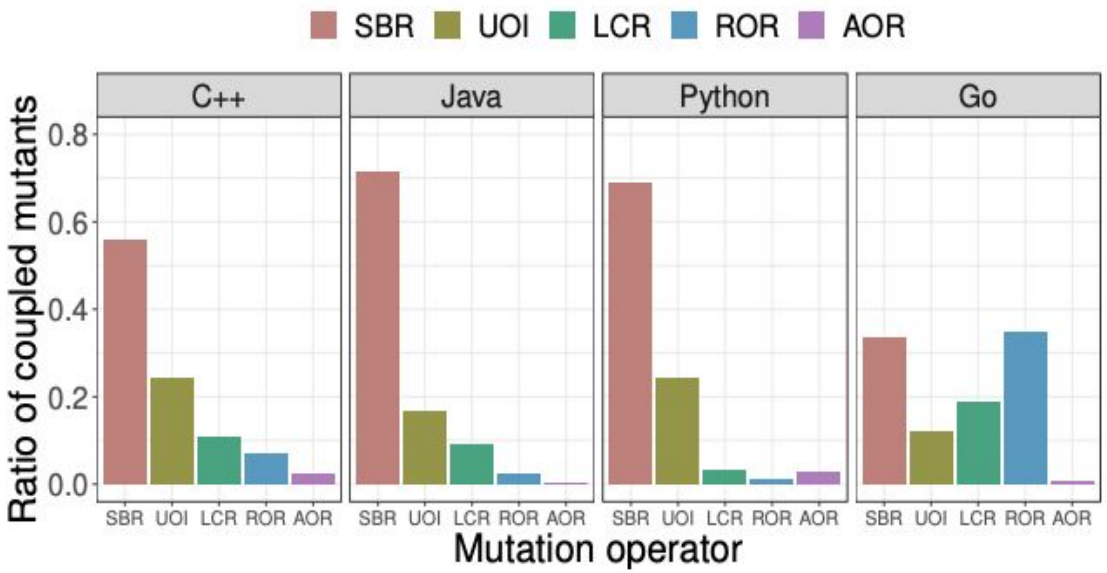
For each bug:

1) Ran dependency analysis and determine all test targets affected by the bug-fixing change.
2) Using the buggy source code version, generate all possible mutants for all lines that are affected by the bug-fixing change.
3) Using the fixed source code version, generate all possible mutants for all lines that are affected by the bug-fixing change.
4) Determine the set of fault-coupled mutants.

Evaluation RQ3

For 1043 (70%) of the bugs, mutation testing would have reported a fault-coupled mutant in the bug-introducing change.

Coupled Faults



The SBR mutation operator generated most of the coupled mutants

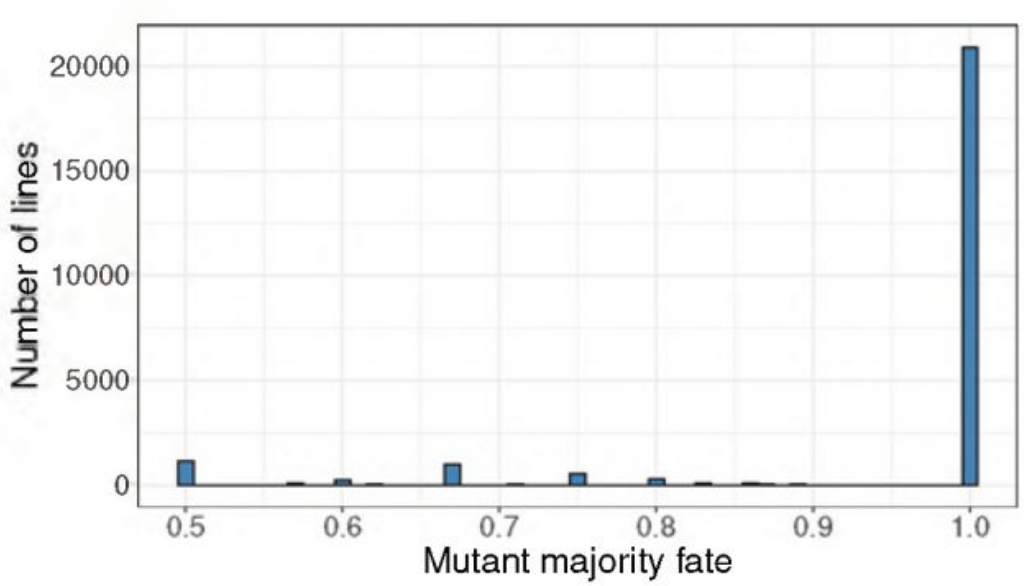
Non-Coupled Faults

30% of bugs were not coupled

- Weak mutation operator
- Missing mutation operator
- No such mutation operator

Evaluation RQ4

Mutants are heavily redundant in 90% of the cases. Either all mutants in a line are killed, or none are killed



- Looked at the testing outcomes of mutants in lines for which multiple mutants were generated.
- Calculated the ratio of the majority event for each line—that is, the ratio of mutants in the majority group sharing the same fate (live or killed).
- This allows them to capture both cases when most mutants are killed and when most mutants are not; the closer the majority fate is to 100%, the higher the redundancy.

Assumptions

- Coupling Effect
  - Simple faults are coupled to more complex faults. Test suite that detects simple faults is sensitive enough to detect complex faults as well.

- Code Author's Push Back
  - Pushback on writing test cases that kill unproductive mutants

- Reviewer's Push Back
  - Pushback on low-quality tests written to kill unproductive mutants (tests which kill only specific mutants that are unproductive but not all mutants)

- Generating Multiple Mutants for the same line(s) of code is not needed
  - Leads to mutant redundancy (if 1 gets killed, all the corresponding also die for the same line)

Novelty

- Prior work assumed that mutants are actionable test goals that improve test suites.
  - This assumption was never validated before, and the authors decided to do so.

- No prior study is found which describes the long-term effects of mutation testing on test quality, quantity, and developer behavior.
  - Authors utilized a big dataset(14 million mutants spanning over 6 years of development) which is novel as compared to conventional work in the domain

- Reports on weather critical real faults are coupled to mutants or not .
  - Not explored in any studies conducted in industrial environments

- Novelty on metrics such as exposure, test hunks, and mutation survivability?

Limitations

- Reviewers' inability to identify low quality test
- Reliance on exposure metric
- Other confounding factors

Practical Significance

- Enhanced Understanding of Mutation Testing Impact
- Informed Decision-Making for Development Teams
- Provides insight on Test Quality and Test Quantity improvement

Discussion points

1. Adoption in software industry
   - Even though the test quality increases with mutation, will more organizations adopt it given the tradeoff of developer's effort and time?

2. Metric to measure developer effort
   - Can the authors include a metrics that measures the developer effort for killing mutant ? Does this help the developers and reviewers make better decisions?

3. Effect on the results reported after inclusion of additional mutation operators
   - One way to do it is to include learned mutation operators from past bug inducing changes
   - can lead to more fault coupled mutants to be reported

4. Limited by reviewer expertise The study could be extended to include a metric to measure reviewer expertise and see the effects of how results change
   - The decision to ask for tests be written or not for a live mutant is dependt on knowledge and the expertise the reviewer possesses