On the Use of Mutation Analysis for Evaluating Student Test Suite Quality

James Perretta[1], Andrew DeOrio[2], Arjun Guha[1], Jonathan Bell[1]

[1]Northeastern University, [2]University of Michigan

Grading Student Test Suites Is Laborious

- Need to teach students how to test software
- Test suite quality metrics can be used as pedagogical tools
- Two typical approaches for grading test suites: both require significant effort

Grading student tests with manually-seeded faults

Instructor seeds faults

Run each student test suite on seeded faults

Grading student tests with “All-Pairs” (requires student implementations, too)

Students provide (possibly buggy) implementations

Run each student test suite on each student implementation

Mutation Analysis Could Emulate These Approaches

- Evidence in prior work that mutants are a valid substitute for faults in OSS
  - Does this generalize to student-written code?
- No prior evidence that mutants are a valid substitute for manually-seeded faults
- Prior work reaches conflicting conclusions on mutation analysis for evaluating student-written tests

An Empirical Study on the Use of Mutation in Grading

- Examine whether mutation analysis is effective way of evaluating student tests
- Large-scale empirical evaluation of student test suites in 2 grading scenarios

RQ1: Is mutation score a good proxy for manually-seeded fault detection rate?

RQ2: Is mutation score a good proxy for faulty student implementation detection rate in an “all-pairs” grading approach?

Empirical Study - Datasets & Tools

- 2,711 assignment submissions total (1 submission/student/assignment)
- Independently-developed impls of the same spec

Mutation Analysis Tools:

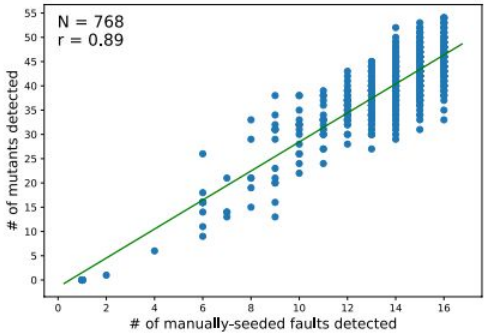
- Stryker Mutator (JS/TS)
- Mull (C++)

Assignment	School	Course	# of Submissions	Has Student Impls	# of Submissions/Day	LOC
Game Card [19]	UMich	EECS 280 [32]	768	Yes	3	136
Game Player [19]	UMich	EECS 280 [32]	762	Yes	3	127
Stable Marriage [19]	UMass	CS220 [30]	485		Unlimited	79
WebApp [19]	Northeastern	CS4530 [31]	93		Unlimited	265
Sorting [19]	Northeastern	CS4530 [31]	90		5	190

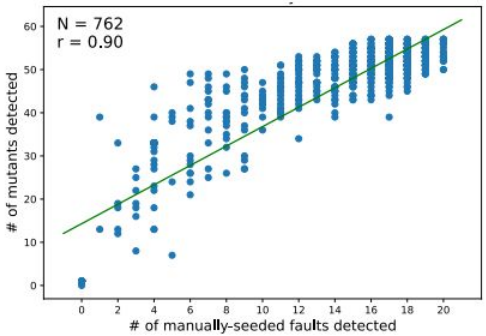
Guarding Against False Positives

- Need to detect test cases that make incorrect assertions
  - Otherwise, a single test with `assert(false);` would incorrectly receive full credit
- Common solution: Run student test suite against one or more correct impls before running it against faulty impls
- Instructors use minor variations on this approach
  - We used the same approach as in the original assignment grading

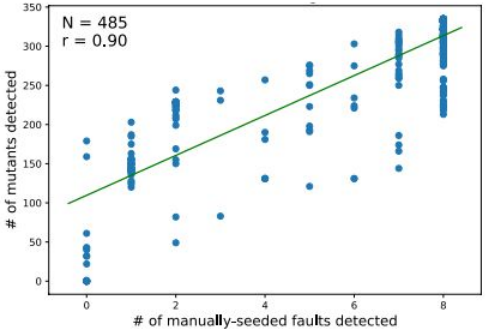
Mutation Detection Is Correlated with Manually-Seeded Fault Detection (RQ1)



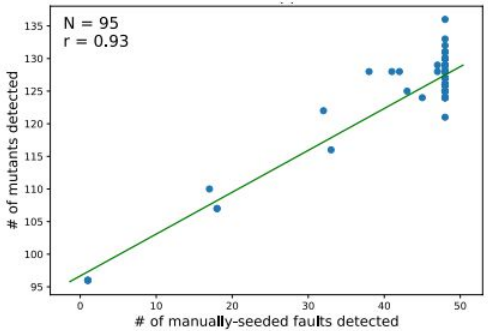
(a) Game Card



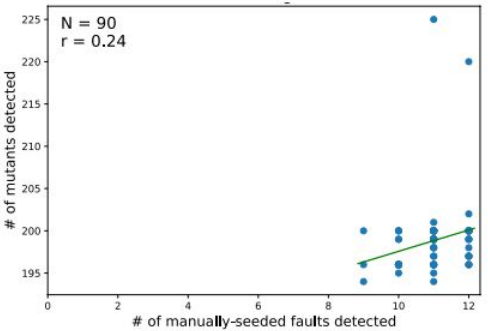
(b) Game Player



(c) Stable Marriage



(d) WebApp



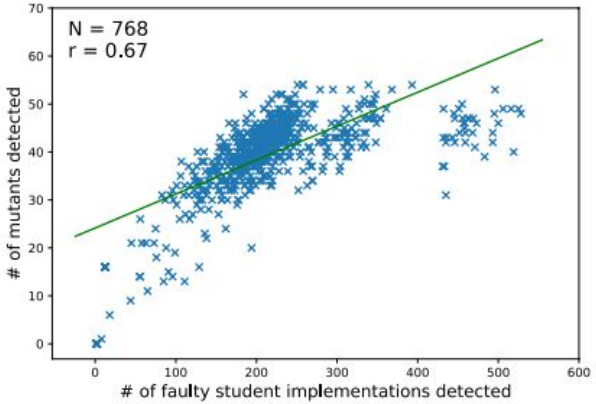
(e) Sorting

Correlation Is Statistically Significant When Controlling for Coverage

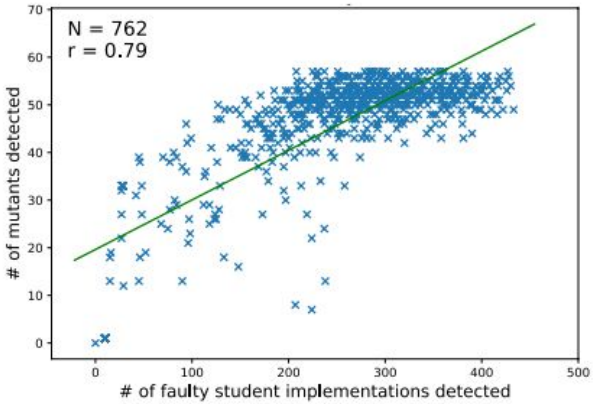
- Control for coverage w/ methodology from Just et al.
- Detecting more manually-seeded faults -> Higher mutation score

Assignment	$(\tilde{T}_{fail}, \tilde{T}_{pass})$ **populations**	**Significant**
Game Card	15/15	15/15
Game Player	20/20	20/20
Stable Marriage	8/8	8/8
WebApp	47/48	47/47
Sorting	8/12	4/8

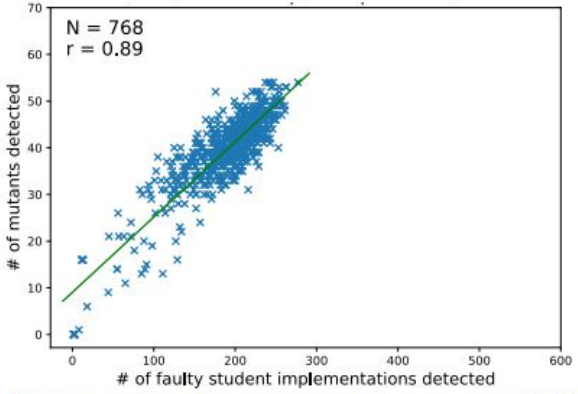
Mutation Detection Is Correlated with “All-Pairs” Faulty Impl Detection (RQ2)



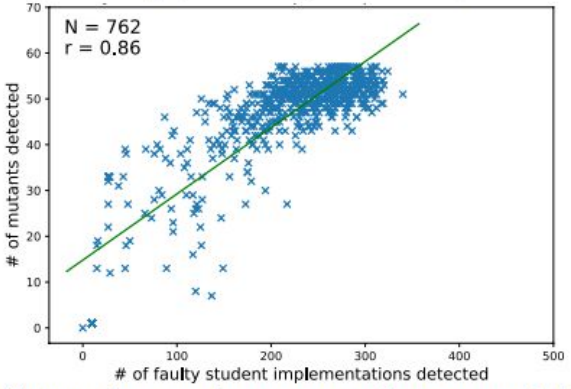
(a) Game Card, considering all student implementations



(b) Game Player, considering all student implementations



(c) Game Card, implementations with mutant-coupled faults



(d) Game Player, implementations with mutant-coupled faults

Not All Faults Are Coupled to Mutants

A few unique student faults were not coupled to a mutant generated by Mull

	Game Card	**Game Player**
Unsupported Mutation Op	5	9
New/Stronger Mutation Op	3	2
Different Impl Structure	7	1
No Mutant	0	14*
Total	15	26

*These faults caused by algorithm flaws, added logic, or incorrect invariant assumptions

Summary of Findings

- Mutants are a very good substitute for manually-seeded faults
  - Strong correlation on 4/5 assignments
  - Avoid low-quality faults like in the Sorting assignment

- Mutants are a reasonably good proxy for all-pairs test suite grading
  - # of redundant mutants doesn’t grow with the # of students (but # of redundant faults in all-pairs does)
  - Correlation weakened by uneven distribution of faults across student impls

Implications

For Researchers:
- Generating mutants from multiple implementations
- Additional research into test suite quality metrics on student test suites

For Educators:
- Consider using mutation analysis tools to generate faults
- Generate mutants from multiple implementations?
- Use mutation analysis outside of the assignment submission loop?

For Tool Builders:
- Better support for educational settings
  - E.g., Support a separate “mutant generation” phase
- Potential use cases for generating mutants from multiple impls or comparing mutation scores of multiple test suites