

Com S 417

Software Testing

Fall 2017 – Week 12, Lecture 21

November 9, 2017

Announcements

- Pezzè and Young, Chapter 16 (Fault-Based Testing) is available on digital reserve.
- Lab 3 grades are posted.
- Lab 5 won't be published until late Friday.

Topics

- Exam 2 Walk Through
- Mutation Testing
 - Weakly vs. Strongly killed mutants
 - The coupling effect.
 - Miscellaneous terminology
 - The mutation test/analysis cycle
 - Relations to structural coverage criteria

More mutants ...

- Stillborn mutants:
 - Dead before the program can be tested (i.e., mutants that produce syntax errors found by the compiler.)
- Trivial mutants:
 - Killed by almost any test
 - For example, a fault that throws a null pointer error in first line
- Distinguished mutants:
 - If “equivalent faults” produce mutants with behavior indistinguishable from the original program, then mutants which can be “killed” must be “distinguished”.

Mutants and test requirements

- Distinguished mutants represent are killed by “useful” tests.
- Equivalent mutants often represent infeasible test cases.

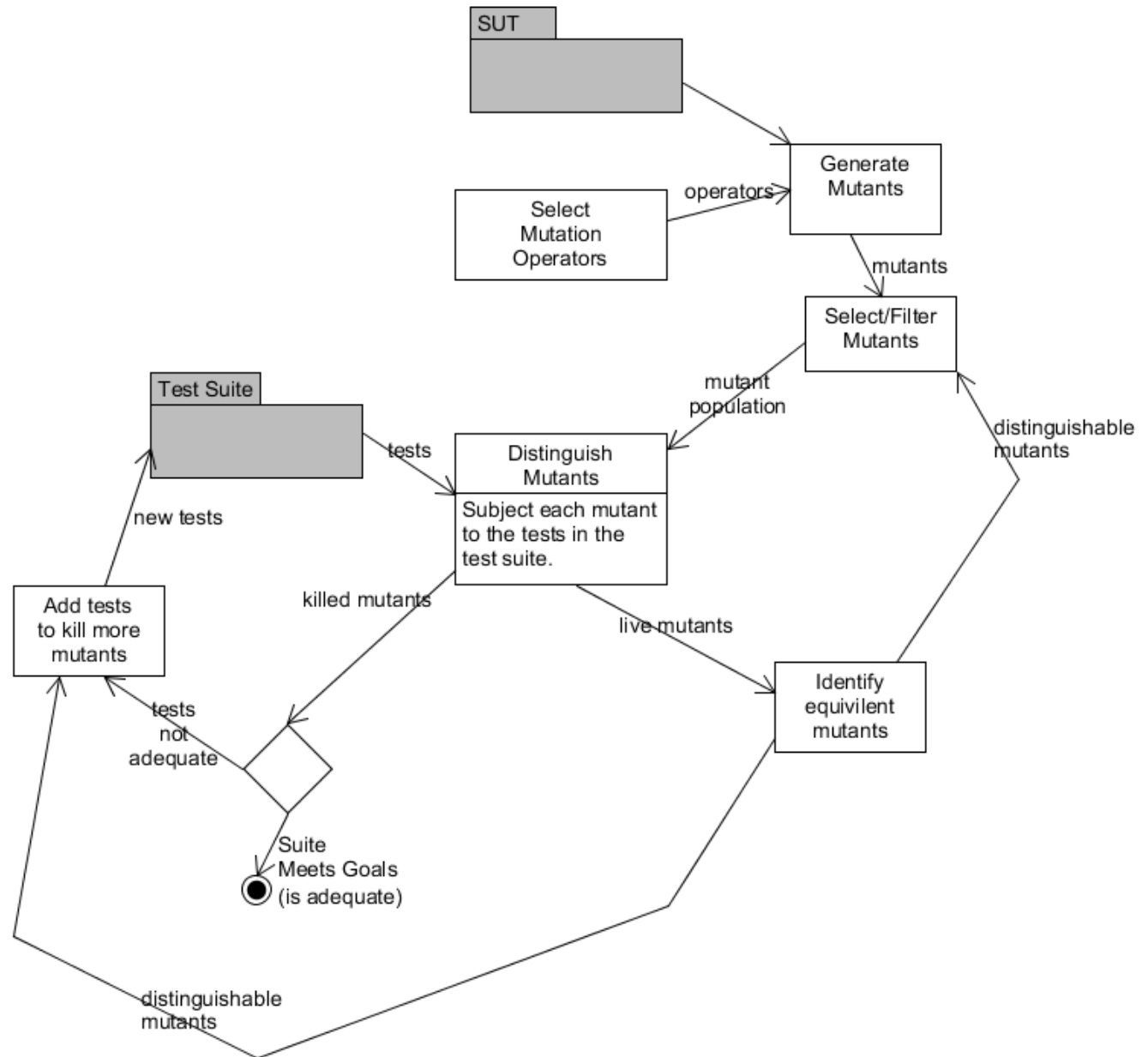
Empirical Results

“For most collections of operators, the number of program-based mutants is roughly proportional to the product of the number of references to variables times the number of variables that are declared.” [Amman & Offutt]

Mutation Testing

Test suite evaluation and improvement.

Slightly different if used for population estimation.



Mutants Vs. Structural Coverage

“For typical sets of syntactic mutants, a mutation-adequate test suite will also be adequate with respect to [criteria such as] statement or branch coverage.”

- Killing a mutant implies you reached the fault.
- If there is a distinguishable mutant for every line of code, then killing all the mutants means you reached every line!
So ... how about an operator that omits individual lines?
- Branch coverage can be “forced” by using an operator that replaces predicates with constant values (true and false).

What about complex interactions?

Coupling Effect

Test data that distinguishes all programs differing from a correct one by only simple errors is so sensitive that it also implicitly distinguishes more complex errors.



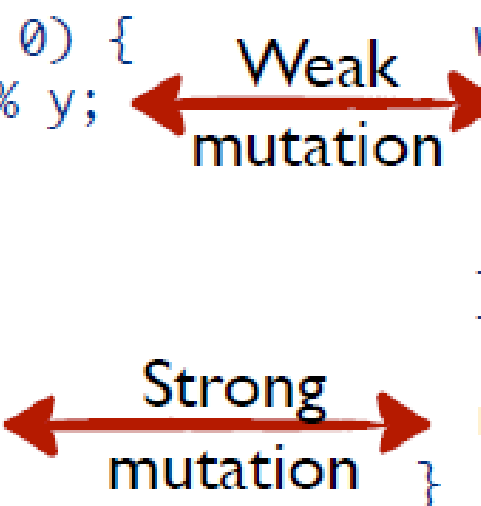
Strong vs. Weak

- Strong mutation (testing to “strongly” kill mutants) requires that the generated fault be reached, cause infection, propagate, and cause a *failure*.
 - This effectively forces each generated fault to “live” in a different mutant – contributing significantly to test execution time.
- Weak mutation only requires the test reach and infect.
 - Since we know where the mutation (fault) is, we can pause execution immediately after the fault is reached to inspect for infection.
 - Reported to save 50% of execution cost. [Fraser]s

Strong requires failure in output.

Weak only requires detectable error in internal state.

Strong vs. Weak Mutation

<pre>int gcd(int x, int y) { int tmp; while(y != 0) { tmp = x % y; x = y; y = tmp; } return x; }</pre>		<pre>int gcd(int x, int y) { int tmp; while(y != 0) { tmp = x * y; x = y; y = tmp; } return x; }</pre>
--	---	--

Weak Mutation Techniques

- Typically depend upon instrumentation injected near the spot of the fault.
- Can inject several faults, each in a different “segment” or “component” of the program. Different data triggers different execution paths.
 - Here “component” is often just an expression or statement of two.
- The savings is primarily in execution time, and compilation time (fewer mutated copies means fewer compiles.)

Weak Mutation

- Offutt and Lee comment:
“To perform weak mutation, the intermediate states of the original program must be saved for each execution of each program component. Since the state contains the entire variable space and additional components, the amount of saved information may be relatively large.”

Weak Mutation Execution Strategy

Again from Offutt and Lee:

"If the entire original program must be executed to record the intermediate states every time a mutant is executed most of the speed advantage of weak mutation will be lost. To reduce this loss, Leonardo executes mutants in order of their comparison points (i.e., the end of the mutated program component). The original program is first executed to a given comparison point, and then all mutants with that comparison point are executed. After these mutants are executed, Leonardo executes the original program to the next comparison point. By doing this, the original program only needs to be executed once for a group of mutants instead of once for each mutant."

Test Set Sizes

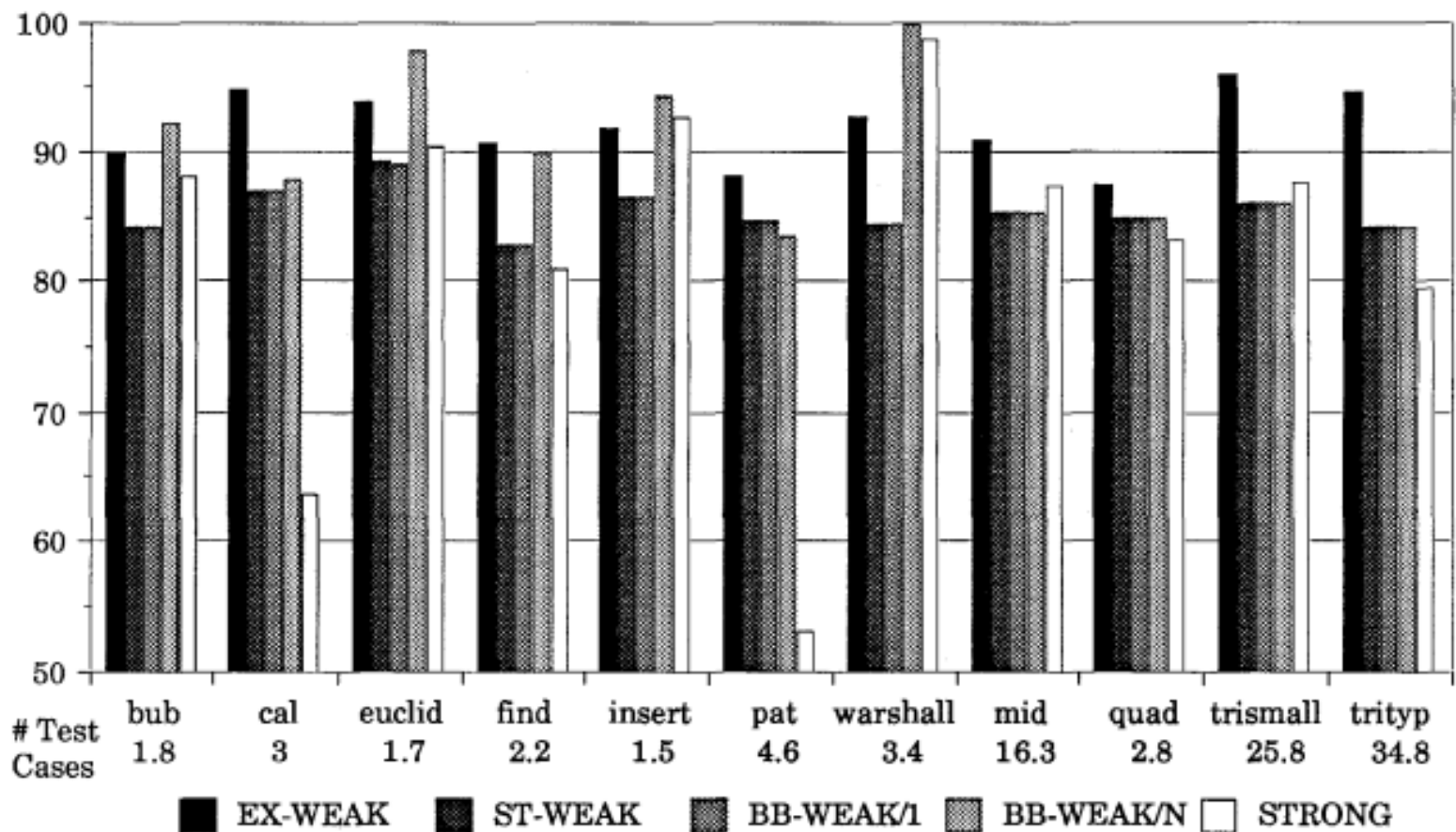


Figure 3: Weak Mutation Variant Comparisons

Mutation Testing Resources

- Ammann & Offutt: 1st ed: Chapter 5 or 2nd ed: Chapter 9
- Pezzè and Young, "Software Testing and analysis: process, principles, and techniques," Chapter 16.
- µJava: <http://cs.gmu.edu/~offutt/mujava/>
- MuClipse: <http://muclipse.sourceforge.net/>
- Offutt and Lee
A. J. Offutt and S. D. Lee, "How strong is weak mutation?," in *Proceedings of the symposium on Testing, analysis, and verification*, 1991, pp. 200–213.