Mull: short overview

Who is Stanislav?

- iOS developer by day, 7 years of experience in software development
- 1.5 years of programming for LLVM in a spare time
- Interests: developer tools, software verification, LLVM
- http://stanislaw.github.io/
- https://systemundertest.org/
- https://github.com/mull-project/mull

What is Mull?

- Mutation testing system based on top of LLVM
- Potentially can work with any programming language that compiles to LLVM IR
- Already implemented:
  - C/C++
  - Rust (on hold, we know it works)
- Pending:
  - Objective-C
  - Swift

Mutation Testing: raw algorithm

```
run_test(program, test);
mutant = mutate(program);
result = run_test(mutant, test)
if (result == Failed) {
  report_killed_mutant(mutant, test);
} else {
  report_survived_mutant(mutant, test);
}
```

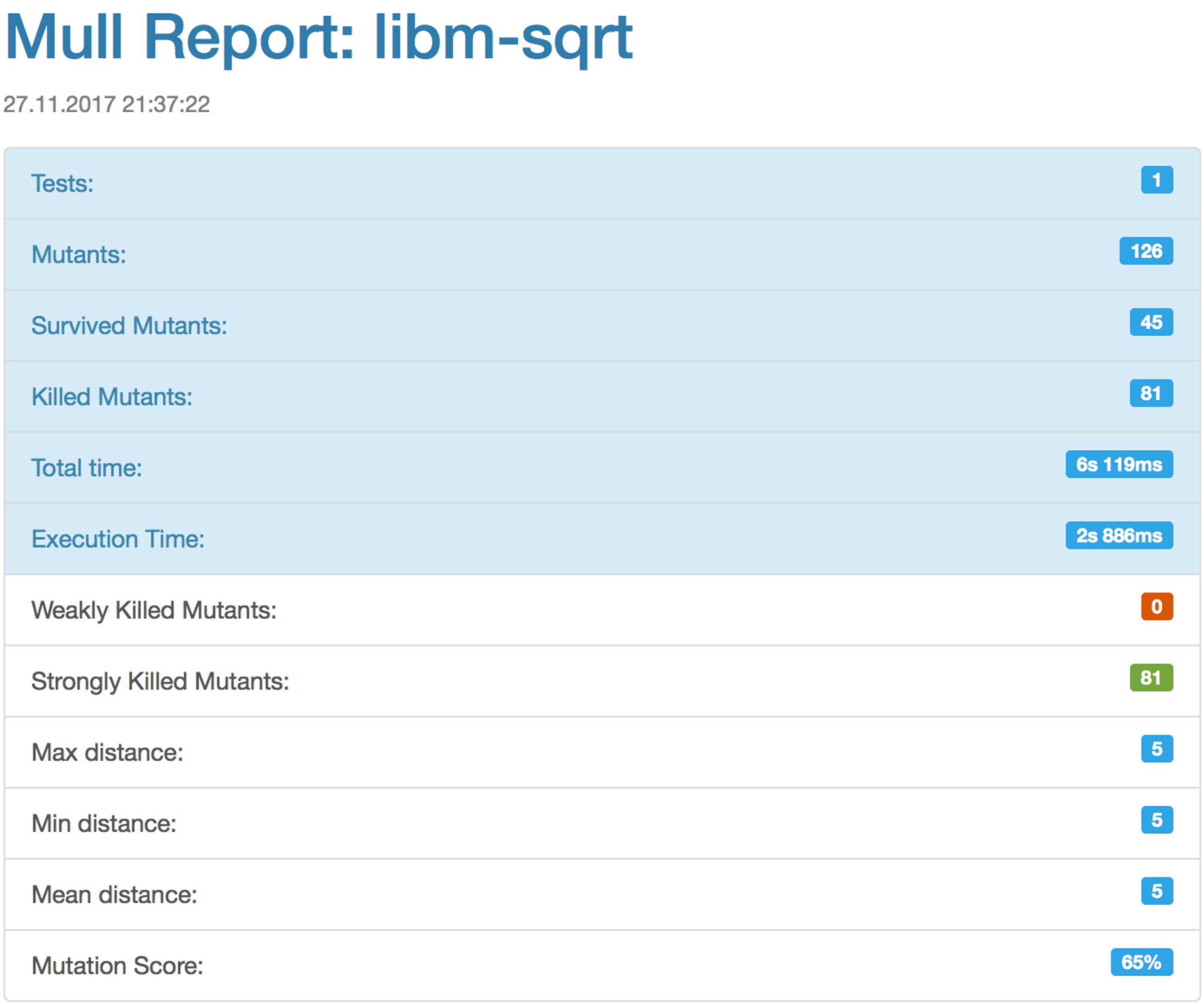
Killed mutant means our test is good: it is able to detect a change in a code.

Survived mutant means our test is not able to detect a change in a code: we either need to improve existing test or write more tests to kill survived mutants.

`assert(sum(10, 5) > 0)` is the example of a shallow test: mutation `+ -> -` will survive. To kill the mutants we just need to make it `assert(sum(10, 5) == 15);`

Demo: sqrt function from newlib/libm library

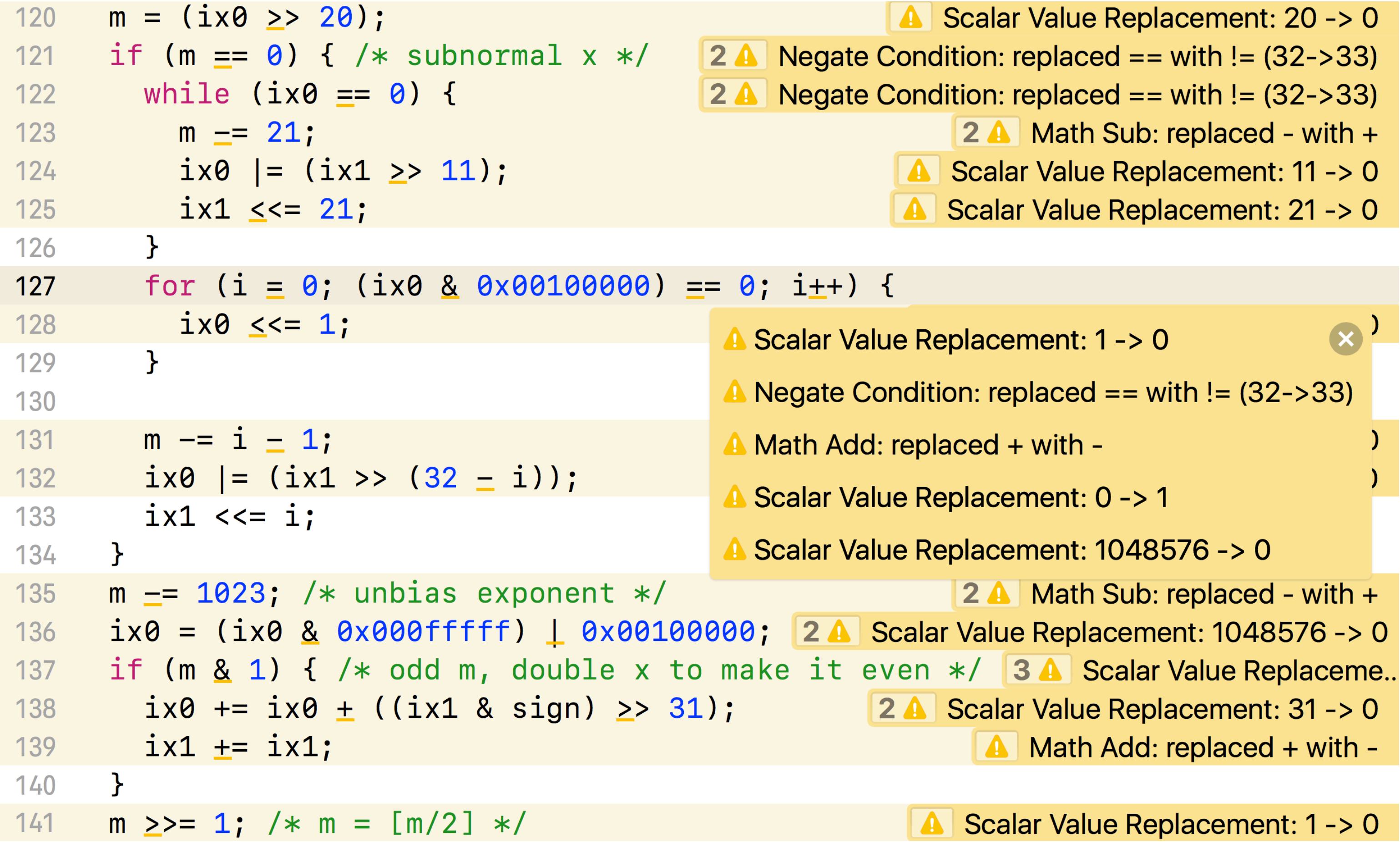
```
#ifdef __STDC__
double __ieee754_sqrt(double x)
#else
double __ieee754_sqrt(x) double x;
#endif
{
  double z;
  __int32_t sign = 0x80000000;
  __uint32_t r, t1, s1, ix1, q1;
  __int32_t ix0, s0, q, m, t, i;

  EXTRACT_WORDS(ix0, ix1, x);

  /* take care of Inf and NaN */
  if ((ix0 & 0x7ff00000) == 0x7ff00000) {
    return x * x + x; /* sqrt(NaN)=NaN, sqrt(+inf)=+inf
                         sqrt(-inf)=sNaN */
  }
  /* take care of zero */
  if (ix0 <= 0) {
    if (((ix0 & (~sign)) | ix1) == 0)
      return x; /* sqrt(+-0) = +-0 */
  ...
  ...
```



Mull Report: libm-sqrt

27.11.2017 21:37:22

Tests:	1
Mutants:	126
Survived Mutants:	45
Killed Mutants:	81
Total time:	6s 119ms
Execution Time:	2s 886ms
Weakly Killed Mutants:	0
Strongly Killed Mutants:	81
Max distance:	5
Min distance:	5
Mean distance:	5
Mutation Score:	65%



```
m = (ix0 >> 20);                                        ⚠ Scalar Value Replacement: 20 -> 0
if (m == 0) { /* subnormal x */                       2 ⚠ Negate Condition: replaced == with != (32->33)
 while (ix0 == 0) {                                   2 ⚠ Negate Condition: replaced == with != (32->33)
  m -= 21;                                            2 ⚠ Math Sub: replaced - with +
  ix0 |= (ix1 >> 11);                                   ⚠ Scalar Value Replacement: 11 -> 0
  ix1 <<= 21;                                           ⚠ Scalar Value Replacement: 21 -> 0
 }
 for (i = 0; (ix0 & 0x00100000) == 0; i++) {
  ix0 <<= 1;                                                                                    )
 }

 m -= i - 1;                                                                                    )
 ix0 |= (ix1 >> (32 - i));                                                                      )
 ix1 <<= i;
}
m -= 1023; /* unbias exponent */                      2 ⚠ Math Sub: replaced - with +
ix0 = (ix0 & 0x000fffff) | 0x00100000;                2 ⚠ Scalar Value Replacement: 1048576 -> 0
if (m & 1) { /* odd m, double x to make it even */    3 ⚠ Scalar Value Replaceme..
 ix0 += ix0 + ((ix1 & sign) >> 31);                   2 ⚠ Scalar Value Replacement: 31 -> 0
 ix1 += ix1;                                            ⚠ Math Add: replaced + with -
}
m >>= 1; /* m = [m/2] */                                ⚠ Scalar Value Replacement: 1 -> 0
```

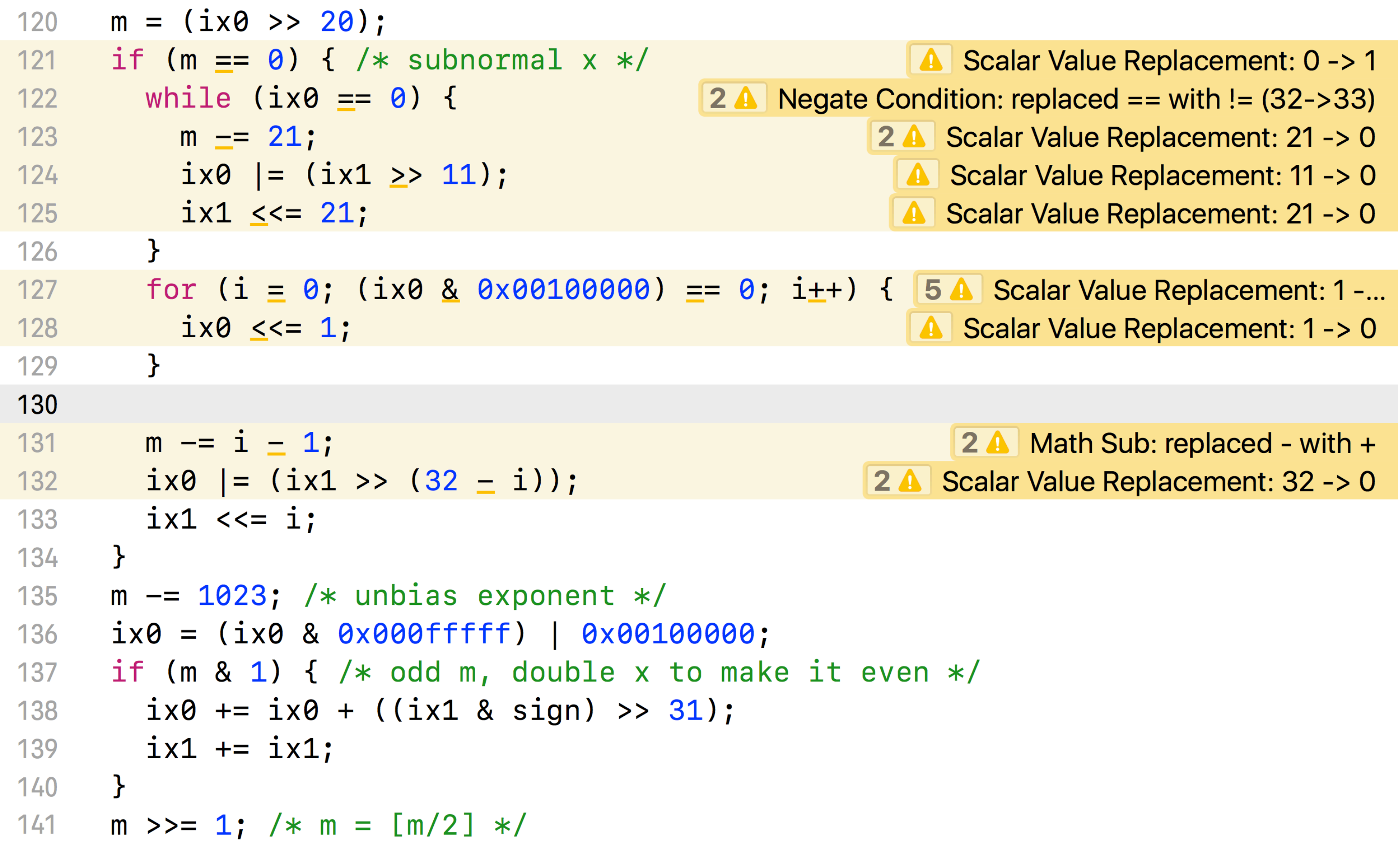
⚠ Scalar Value Replacement: 1 -> 0 ✕

⚠ Negate Condition: replaced == with != (32->33)

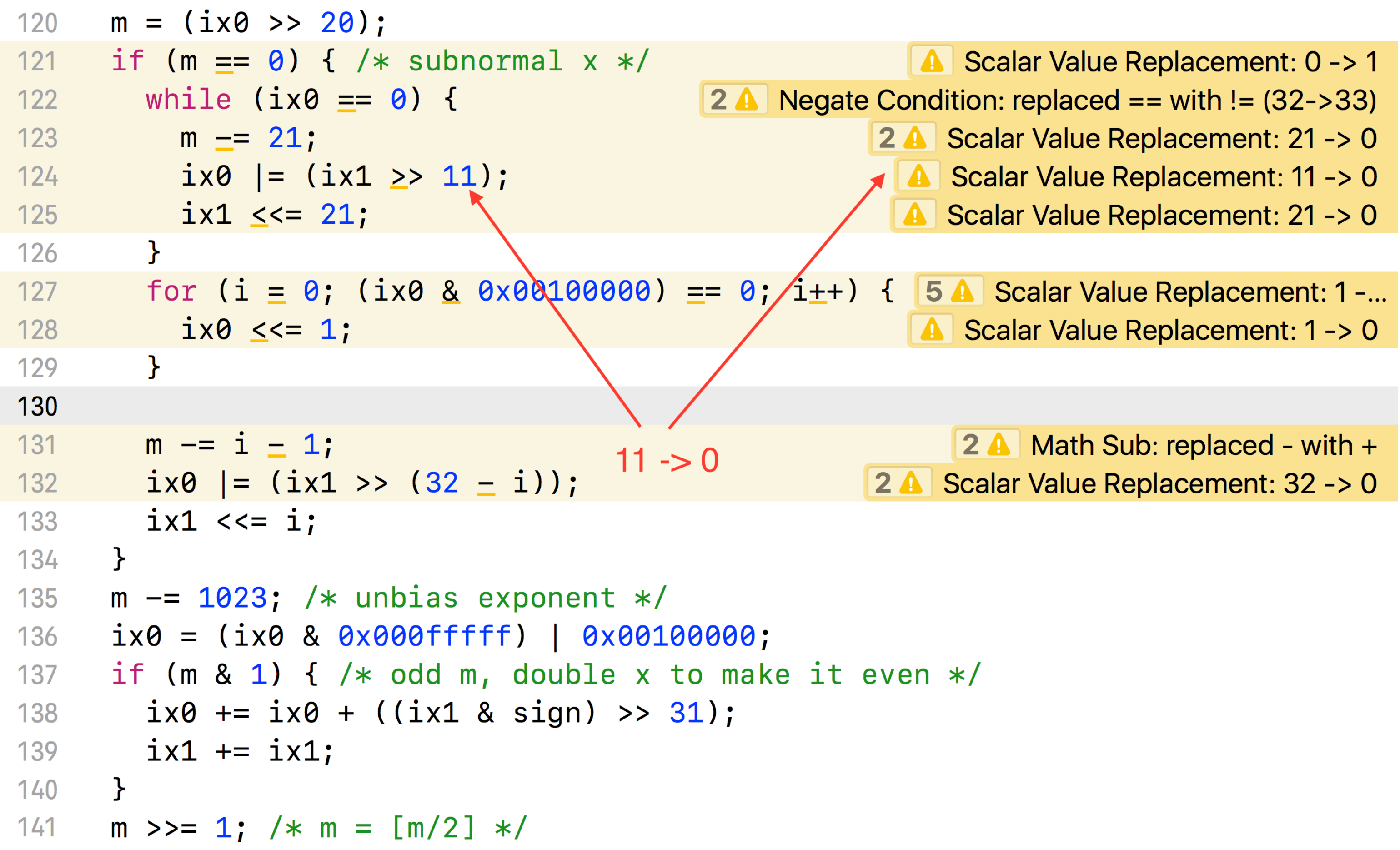
⚠ Math Add: replaced + with -

⚠ Scalar Value Replacement: 0 -> 1

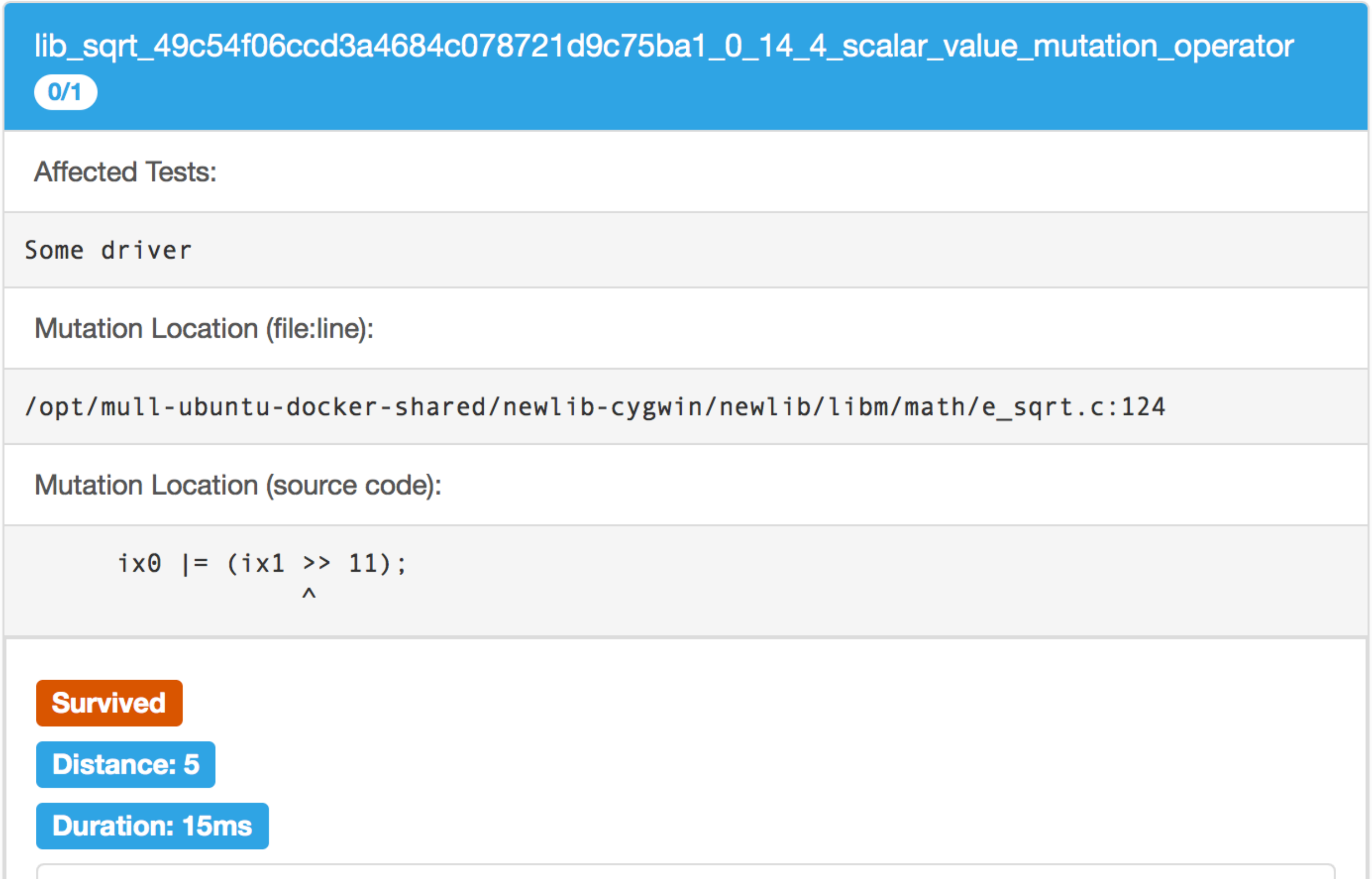
⚠ Scalar Value Replacement: 1048576 -> 0



```
m = (ix0 >> 20);
if (m == 0) { /* subnormal x */                                ⚠ Scalar Value Replacement: 0 -> 1
  while (ix0 == 0) {                     2 ⚠ Negate Condition: replaced == with != (32->33)
    m -= 21;                                                 2 ⚠ Scalar Value Replacement: 21 -> 0
    ix0 |= (ix1 >> 11);                                        ⚠ Scalar Value Replacement: 11 -> 0
    ix1 <<= 21;                                                ⚠ Scalar Value Replacement: 21 -> 0
  }
  for (i = 0; (ix0 & 0x00100000) == 0; i++) {   5 ⚠ Scalar Value Replacement: 1 -...
    ix0 <<= 1;                                                 ⚠ Scalar Value Replacement: 1 -> 0
  }

  m -= i - 1;                                                  2 ⚠ Math Sub: replaced - with +
  ix0 |= (ix1 >> (32 - i));                                  2 ⚠ Scalar Value Replacement: 32 -> 0
  ix1 <<= i;
}
m -= 1023; /* unbias exponent */
ix0 = (ix0 & 0x000fffff) | 0x00100000;
if (m & 1) { /* odd m, double x to make it even */
  ix0 += ix0 + ((ix1 & sign) >> 31);
  ix1 += ix1;
}
m >>= 1; /* m = [m/2] */
```



```
m = (ix0 >> 20);
if (m == 0) { /* subnormal x */                          ⚠ Scalar Value Replacement: 0 -> 1
  while (ix0 == 0) {                           2 ⚠ Negate Condition: replaced == with != (32->33)
    m -= 21;                                            2 ⚠ Scalar Value Replacement: 21 -> 0
    ix0 |= (ix1 >> 11);                                   ⚠ Scalar Value Replacement: 11 -> 0
    ix1 <<= 21;                                           ⚠ Scalar Value Replacement: 21 -> 0
  }
  for (i = 0; (ix0 & 0x00100000) == 0; i++) {        5 ⚠ Scalar Value Replacement: 1 -...
    ix0 <<= 1;                                            ⚠ Scalar Value Replacement: 1 -> 0
  }

  m -= i - 1;                                            2 ⚠ Math Sub: replaced - with +
  ix0 |= (ix1 >> (32 - i));                             2 ⚠ Scalar Value Replacement: 32 -> 0
  ix1 <<= i;
}
m -= 1023; /* unbias exponent */
ix0 = (ix0 & 0x000fffff) | 0x00100000;
if (m & 1) { /* odd m, double x to make it even */
  ix0 += ix0 + ((ix1 & sign) >> 31);
  ix1 += ix1;
}
m >>= 1; /* m = [m/2] */
```



lib_sqrt_49c54f06ccd3a4684c078721d9c75ba1_0_14_4_scalar_value_mutation_operator

0/1

Affected Tests:

```
Some driver
```

Mutation Location (file:line):

```
/opt/mull-ubuntu-docker-shared/newlib-cygwin/newlib/libm/math/e_sqrt.c:124
```

Mutation Location (source code):

```
      ix0 |= (ix1 >> 11);
                    ^
```

Survived

Distance: 5

Duration: 15ms

How to make Mull support a programming language?

- Language must be LLVM-friendly
    - Should support compilation to LLVM IR: `*.bc` and `*.ll`
- Find tests
    - Each programming language has its own implementation details.
    - Algorithm depends on a test framework used: Google Test, XCTest, Rust's native test framework etc.
- Find testees (code that is tested), find mutation points in them
- Make tests run with LLVM JIT

How to run Mull on a project

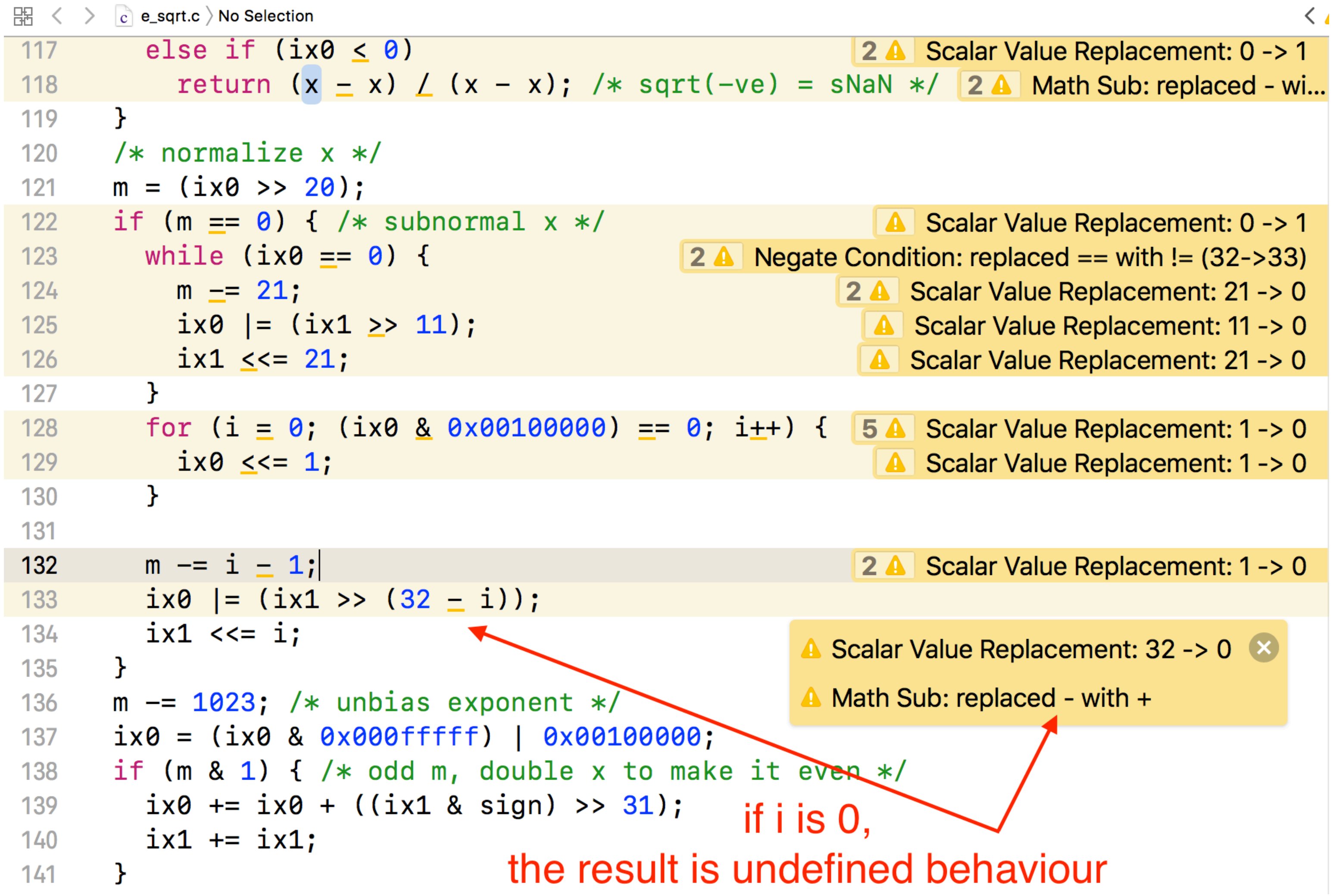
- Step 1: Building Mull
- Step 2: Getting LLVM bitcode
- Step 3: Creating config.yml file
- Step 4: Running Mull
- Step 5: Generating HTML report

Mutation Testing Coverage and Code Coverage

- Statement coverage < mutation testing coverage
- Branch coverage <= mutation testing coverage
  - Negate condition mutation operator: `true <-> false`
- Condition coverage <= mutation testing coverage
  - Negate condition mutation operator: `true <-> false`
  - AND-OR mutation operator: `&& <-> ||`
- MC/DC coverage > mutation testing coverage
  - Tried NASA tutorial: Mull gives 100% coverage on examples that do not satisfy MC/DC.
- Mutation testing is not the best coverage but it is much better than statement coverage. We see it as a good replacement for statement coverage to raise the standards of testing.

Mutation Testing and Symbolic Execution

- Symbolic execution is an extremely friendly technique
- KLEE is a tool for symbolic execution, also LLVM-based
- KLEE can generate tests automatically
- "Mull and Klee, part 1: mutation testing analysis for Klee's Tutorial Two"
    - Compared Mull and KLEE on a simple C function. KLEE generated tests with maximum of 87% mutation coverage.
    - Test-generation very much depends on a solver KLEE uses.
    - Needs a human to analyze the results.
    - KLEE cannot generate tests easily readable by human (we reported this: https://github.com/klee/klee/issues/648)



e_sqrt.c 〉 No Selection

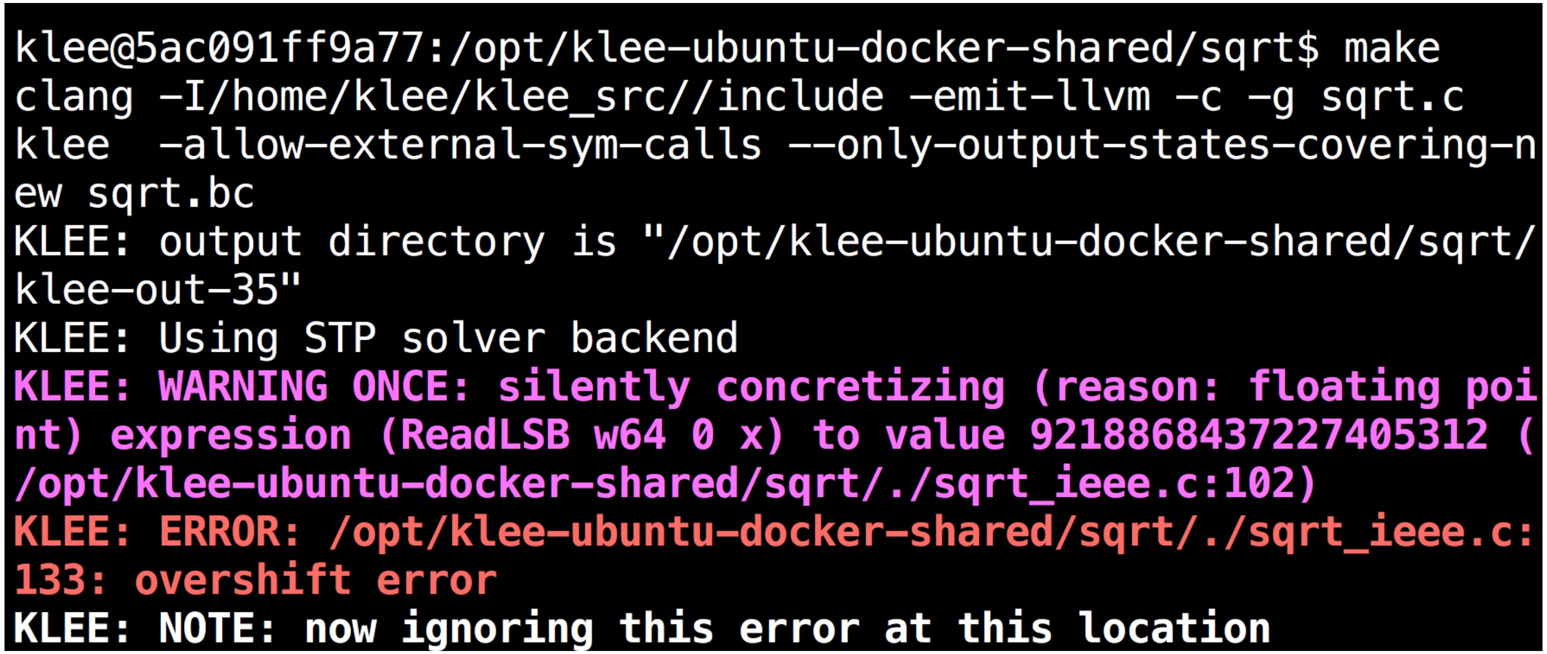
```
    else if (ix0 < 0)
      return (x - x) / (x - x); /* sqrt(-ve) = sNaN */
}
/* normalize x */
m = (ix0 >> 20);
if (m == 0) { /* subnormal x */
  while (ix0 == 0) {
    m -= 21;
    ix0 |= (ix1 >> 11);
    ix1 <<= 21;
  }
  for (i = 0; (ix0 & 0x00100000) == 0; i++) {
    ix0 <<= 1;
  }

  m -= i - 1;
  ix0 |= (ix1 >> (32 - i));
  ix1 <<= i;
}
m -= 1023; /* unbias exponent */
ix0 = (ix0 & 0x000fffff) | 0x00100000;
if (m & 1) { /* odd m, double x to make it even */
  ix0 += ix0 + ((ix1 & sign) >> 31);
  ix1 += ix1;
}
```

- 2 ⚠ Scalar Value Replacement: 0 -> 1
- 2 ⚠ Math Sub: replaced - wi...
- ⚠ Scalar Value Replacement: 0 -> 1
- 2 ⚠ Negate Condition: replaced == with != (32->33)
- 2 ⚠ Scalar Value Replacement: 21 -> 0
- ⚠ Scalar Value Replacement: 11 -> 0
- ⚠ Scalar Value Replacement: 21 -> 0
- 5 ⚠ Scalar Value Replacement: 1 -> 0
- ⚠ Scalar Value Replacement: 1 -> 0
- 2 ⚠ Scalar Value Replacement: 1 -> 0

⚠ Scalar Value Replacement: 32 -> 0

⚠ Math Sub: replaced - with +

if i is 0,
the result is undefined behaviour

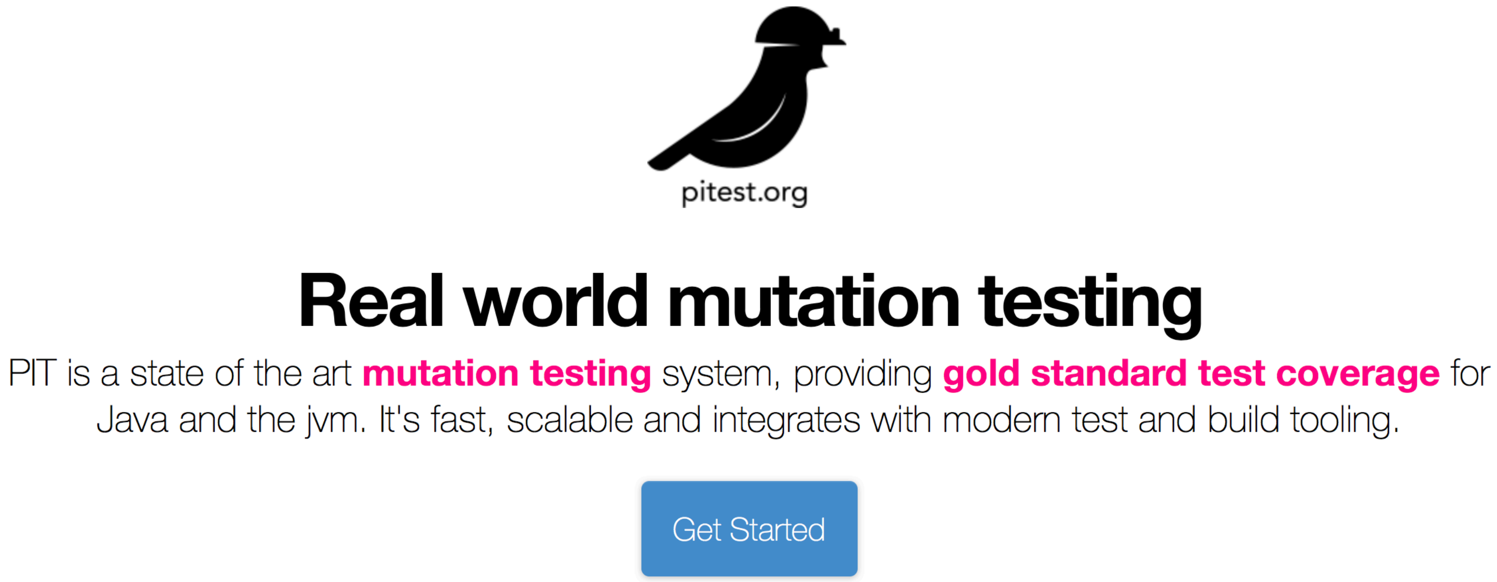


```
klee@5ac091ff9a77:/opt/klee-ubuntu-docker-shared/sqrt$ make
clang -I/home/klee/klee_src//include -emit-llvm -c -g sqrt.c
klee  -allow-external-sym-calls --only-output-states-covering-n
ew sqrt.bc
KLEE: output directory is "/opt/klee-ubuntu-docker-shared/sqrt/
klee-out-35"
KLEE: Using STP solver backend
KLEE: WARNING ONCE: silently concretizing (reason: floating poi
nt) expression (ReadLSB w64 0 x) to value 9218868437227405312 (
/opt/klee-ubuntu-docker-shared/sqrt/./sqrt_ieee.c:102)
KLEE: ERROR: /opt/klee-ubuntu-docker-shared/sqrt/./sqrt_ieee.c:
133: overshift error
KLEE: NOTE: now ignoring this error at this location
```

Mull 2016-2017: brief history

- Proof of concept: running `a + b` then `a - b` using LLVM JIT
- Running tests in a fork'ed process
- Reverse engineering Google Test to run it with LLVM JIT
- C++ support
- Implementation of math: add, sub, mul, div, scalar value, replace call, remove void call, negate condition, AND-OR
- HTML reporting
- Initial Rust support
- Getting to work on Linux Ubuntu and CentOS
- From static test finder to dynamic test finder: find mutations in function pointers and polymorphic classes
- IDE diagnostics
- Analysis of LLVM ADT and Support, fmt, OpenSSL, newlibm/libm libraries

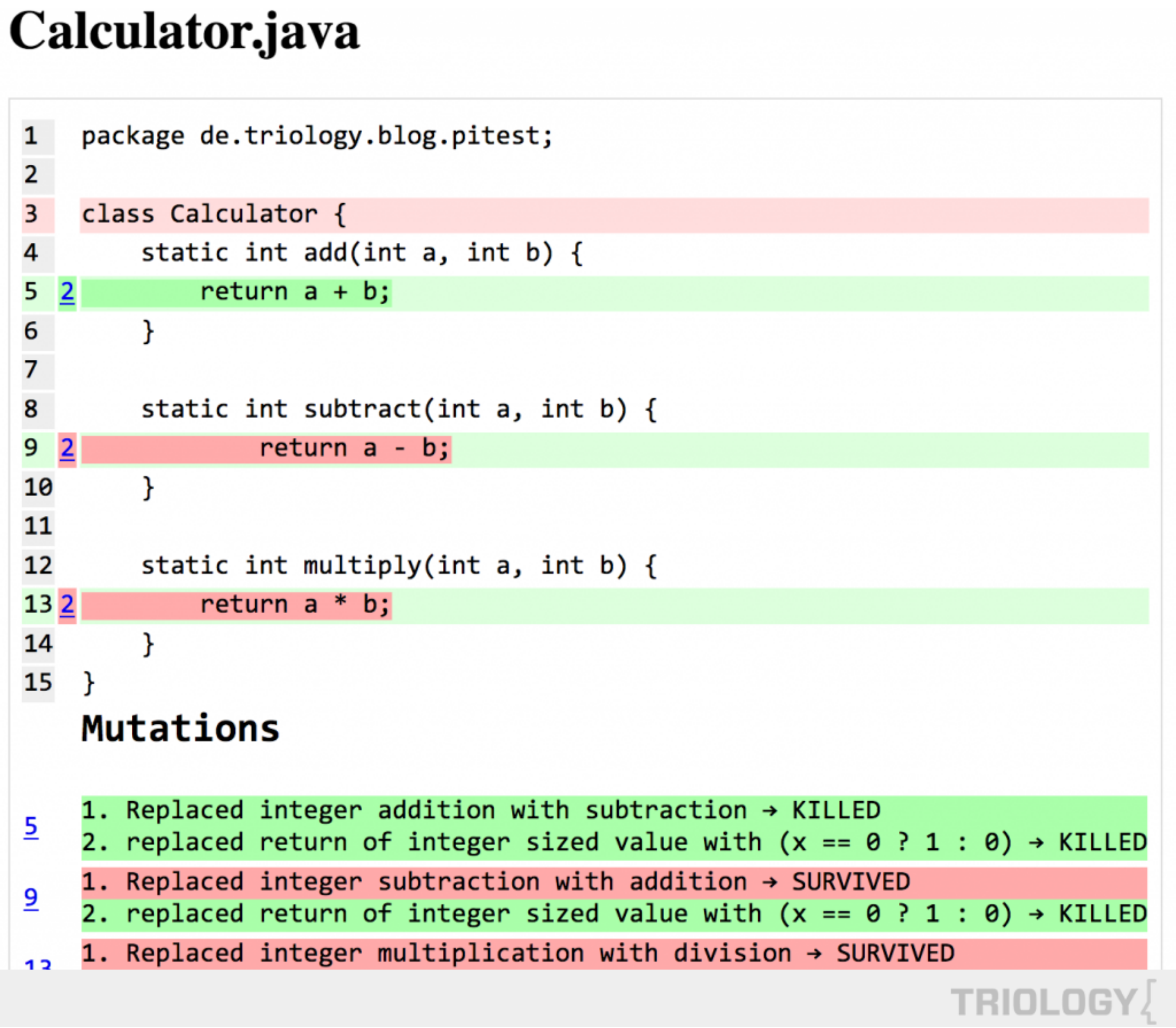
Pitest: examples



Real world mutation testing

PIT is a state of the art **mutation testing** system, providing **gold standard test coverage** for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling.

http://pitest.org



Calculator.java

```
1     package de.triology.blog.pitest;
2
3     class Calculator {
4         static int add(int a, int b) {
5  2          return a + b;
6         }
7
8         static int subtract(int a, int b) {
9  2          return a - b;
10        }
11
12        static int multiply(int a, int b) {
13 2          return a * b;
14        }
15    }
```

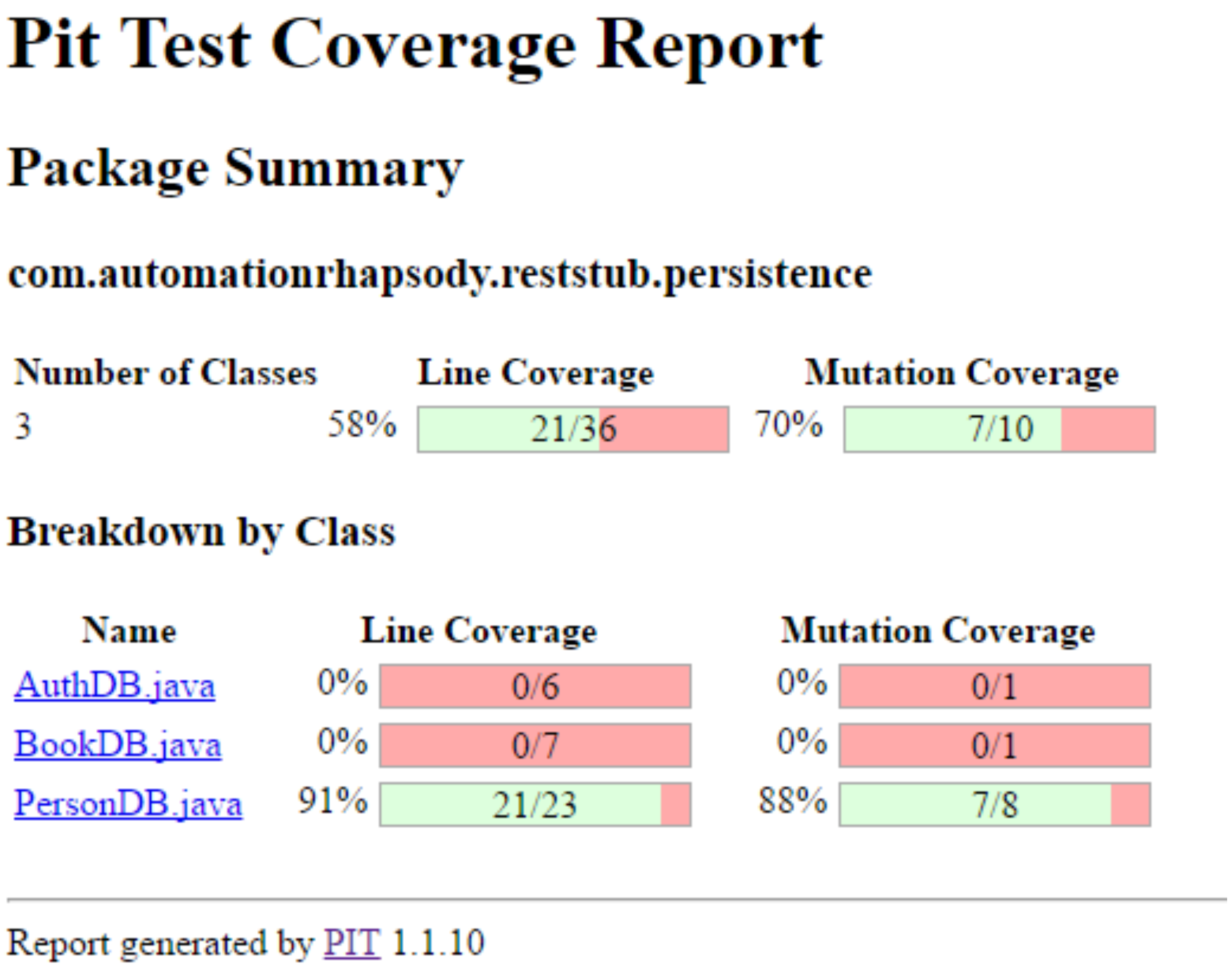
Mutations

5
1. Replaced integer addition with subtraction → KILLED
2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

9
1. Replaced integer subtraction with addition → SURVIVED
2. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

13
1. Replaced integer multiplication with division → SURVIVED

https://www.triology.de/en/blog-entries/mutation-testing



Pit Test Coverage Report

Package Summary

com.automationrhapsody.reststub.persistence

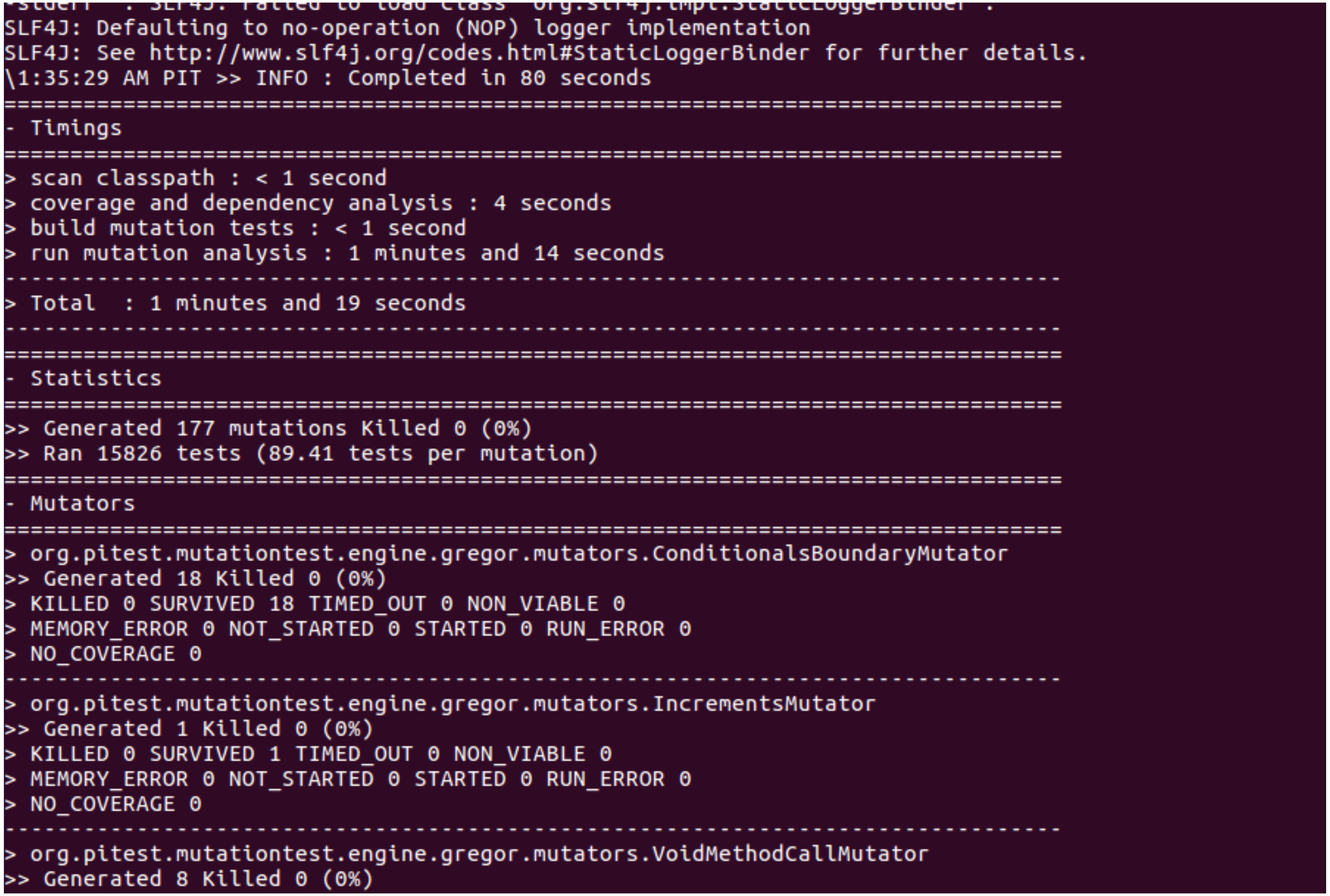
Number of Classes	Line Coverage		Mutation Coverage	
3	58%	21/36	70%	7/10

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
AuthDB.java	0%	0/6	0%	0/1
BookDB.java	0%	0/7	0%	0/1
PersonDB.java	91%	21/23	88%	7/8

Report generated by PIT 1.1.10

https://automationrhapsody.com/mutation-testing-java-pitest/



```
-stderr  : SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
\1:35:29 AM PIT >> INFO : Completed in 80 seconds
================================================================================
- Timings
================================================================================
> scan classpath : < 1 second
> coverage and dependency analysis : 4 seconds
> build mutation tests : < 1 second
> run mutation analysis : 1 minutes and 14 seconds
--------------------------------------------------------------------------------
> Total  : 1 minutes and 19 seconds
--------------------------------------------------------------------------------
================================================================================
- Statistics
================================================================================
>> Generated 177 mutations Killed 0 (0%)
>> Ran 15826 tests (89.41 tests per mutation)
================================================================================
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutator
>> Generated 18 Killed 0 (0%)
> KILLED 0 SURVIVED 18 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 8 Killed 0 (0%)
```

https://www.bountysource.com/teams/pitest/issues?tracker_ids=832315

Further work

High-level goals

- Make Mull a general purpose tool for mutation testing.

Implementation

- Much better visual reporting.
- Integration to IDEs.
- Support of Swift / Objective-C
- 1-click integration of Mull into C++/CMake-based projects.

Research

- Auto-generation of tests
  - KLEE and libFuzzer can help

Open questions

These are the questions we want to answer with Mull eventually.

- Can mutation testing actually detect serious errors in the real-world programs?
- What are the most effective mutation operators?
- Compare mutation testing with other methods like static analysis, fuzz testing, etc.
- Can mutation testing help to find errors that a normal testing with a code coverage cannot find?
- Can mutation testing help in automatic test generation?

Questions?