Rotten Green Unit Tests

LLVM Developers' Meeting, November 2021

Paul T. Robinson

Sony Interactive Entertainment

Inspiration

DOI 10.1109/ICSE.2019.00062

2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

Rotten Green Tests

Julien Delplanque*, Stéphane Ducasse†, Guillermo Polito*, Andrew P. Black§† and Anne Etien*
*Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France
†RMOD - Inria Lille, France
§Dept of Computer Science, Portland State University, Oregon, USA
*†{firstname}.{lastname}@inria.fr §apblack@pdx.edu

Testing in Clang and LLVM

An Incomplete Taxonomy

Internal self-testing

- `assert` (arguable)
- `clang -verify`
- IR verifier

Specialized tools

- `c-index-test`
- `opt, llc, llvm-mc`
- `lit, FileCheck`

Executable API tests

- `clang/unittests`
- `llvm/unittests`

End-to-end tests

- `test-suite`
- `cross-project-tests`

Downstream project tests

- Chrome, many more
- Vendors

Exploratory testing

- `godbolt.org`

End users

Testing in Clang and LLVM

An Incomplete Taxonomy

Internal self-testing

- `assert` (arguable)
- `clang -verify`
- IR verifier

Specialized tools

- `c-index-test`
- `opt, llc, llvm-mc`
- `lit, FileCheck`

Executable API tests

- `clang/unittests`
- `llvm/unittests`

End-to-end tests

- `test-suite`
- `cross-project-tests`

Downstream project tests

- Chrome, many more
- Vendors

Exploratory testing

- `godbolt.org`

End users

Unit Testing in Clang and LLVM

Executable Programs to Exercise APIs

- `clang/unittests` and `llvm/unittests`
- Organized by library (e.g., `llvm/unittests/IR`, `clang/unittests/Sema`)

Based on the `googletest` Open-Source Framework

- On GitHub: https://github.com/google/googletest
- User's guide: https://google.github.io/googletest/

Subset (Just Source) of `googletest` Imported to LLVM As Needed

- `llvm/utils/unittest`
- `googletest`'s own test suite is not imported

Some Local Patches Added

- Support for NetBSD, Minix, Haiku
- StringRef and raw_os_ostream support

Unit Test General Structure

A test *fixture*

- Sets up the (sub-)system environment to be tested

One or more *stimuli*

- Exercise the component under test

One or more *test assertions*

- Verify some expected property

(From `llvm/unittests/ADT/SmallSetTest.cpp`)

```
TEST(SmallSetTest, Insert) {
  SmallSet<int, 4> s1;
  for (int i = 0; i < 4; i++)
    s1.insert(i);

  for (int i = 0; i < 4; i++)
    s1.insert(i);

  EXPECT_EQ(4u, s1.size());
  for (int i = 0; i < 4; i++)
    EXPECT_EQ(1u, s1.count(i));
  EXPECT_EQ(0u, s1.count(4));

}
```

Unit Test General Structure

A test *fixture*

- Sets up the (sub-)system environment to be tested

One or more *stimuli*

- Exercise the component under test

One or more *test assertions*

- Verify some expected property

(From `llvm/unittests/ADT/SmallSetTest.cpp`)

```
TEST(SmallSetTest, Insert) {
  SmallSet<int, 4> s1;
  for (int i = 0; i < 4; i++)
    s1.insert(i);

  for (int i = 0; i < 4; i++)
    s1.insert(i);

  EXPECT_EQ(4u, s1.size());
  for (int i = 0; i < 4; i++)
    EXPECT_EQ(1u, s1.count(i));
  EXPECT_EQ(0u, s1.count(4));

}
```

Unit Test General Structure

A test *fixture*

- Sets up the (sub-)system environment to be tested

One or more *stimuli*

- Exercise the component under test

One or more *test assertions*

- Verify some expected property

(From `llvm/unittests/ADT/SmallSetTest.cpp`)

```
TEST(SmallSetTest, Insert) {
  SmallSet<int, 4> s1;
  for (int i = 0; i < 4; i++)
    s1.insert(i);

  for (int i = 0; i < 4; i++)
    s1.insert(i);

  EXPECT_EQ(4u, s1.size());
  for (int i = 0; i < 4; i++)
    EXPECT_EQ(1u, s1.count(i));
  EXPECT_EQ(0u, s1.count(4));

}
```

What is a Rotten Green Test?

A *test* verifies some aspect of the behavior of the software

- Set up some environment/input
- Run (some part of) the software being tested
- Validate the result against some oracle

What is a Rotten Green Test?

A *test* verifies some aspect of the behavior of the software

- Set up some environment/input
- Run (some part of) the software being tested
- Validate the result against some oracle

A *green test* is one that passes

What is a Rotten Green Test?

A *test* verifies some aspect of the behavior of the software

- Set up some environment/input
- Run (some part of) the software being tested
- Validate the result against some oracle

A *green test* is one that ~~passes~~ doesn't fail

- The default condition is taken to be passing

Program testing can be used to show the presence of bugs, but never to show their absence!

--Edsger W Dijkstra, *Notes on Structured Programming*

What is a Rotten Green Test?

A *test* verifies some aspect of the behavior of the software

- Set up some environment/input
- Run (some part of) the software being tested
- Validate the result against some oracle

A *green test* is one that ~~passes~~ doesn't fail

- The default condition is taken to be passing

A *rotten green test* is one that doesn't actually validate the result (correctly)

- Contains assertions that *look like* they validate the result
- But these assertions aren't executed (written incorrectly)
  - Incorrect assertions don't necessarily fail!

Example Rotten Test Assertion

llvm/unittests/ProfileData/CoverageMappingTest.cpp

```
for (const auto &Group : InstantiationGroups)
  ASSERT_EQ(Group.size(), 1U);
```

Example Rotten Test Assertion

llvm/unittests/ProfileData/CoverageMappingTest.cpp

```
for (const auto &Group : InstantiationGroups)
  ASSERT_EQ(Group.size(), 1U);
```

`InstantiationGroups` is invariably empty, so the assertion is never executed. It might *look* useful, but it's rotten.

And if you're reading the test to try to understand how the APIs behave, it is likely to be misleading or confusing.

Example Rotten Test Assertion

llvm/unittests/ProfileData/CoverageMappingTest.cpp

```
for (const auto &Group : InstantiationGroups)
  ASSERT_EQ(Group.size(), 1U);
```

`InstantiationGroups` is invariably empty, so the assertion is never executed. It might *look* useful, but it's rotten.

And if you're reading the test to try to understand how the APIs behave, it is likely to be misleading or confusing.

Fixed in D95258

```
ASSERT_TRUE(InstantiationGroups.empty());
```

Detecting Rotten Test Assertions

Statically Identify All Assertions

- Instrument the `EXPECT/ASSERT` macros
- Record source location and an “Executed” flag
- Associate assertions with containing test methods

Detecting Rotten Test Assertions

Statically Identify All Assertions

- Instrument the `EXPECT/ASSERT` macros
- Record source location and an “Executed” flag
- Associate assertions with containing test methods

Dynamically Record Executed Assertions

- Set the “Executed” flag when the assertion is executed

Detecting Rotten Test Assertions

Statically Identify All Assertions

- Instrument the `EXPECT/ASSERT` macros
- Record source location and an “Executed” flag
- Associate assertions with containing test methods

Dynamically Record Executed Assertions

- Set the “Executed” flag when the assertion is executed

Eliminate Duplicates and False Positives

- Duplicates occur due to template functions (each instantiation will have the same source location)
  - Also with nested macros in gcc
- False positives occur when a test is Disabled, Skipped, or Filtered

Detecting Rotten Test Assertions

Statically Identify All Assertions

- Instrument the `EXPECT/ASSERT` macros
- Record source location and an “Executed” flag
- Associate assertions with containing test methods

Dynamically Record Executed Assertions

- Set the “Executed” flag when the assertion is executed

Eliminate Duplicates and False Positives

- Duplicates occur due to template functions (each instantiation will have the same source location)
  - Also with nested macros in gcc
- False positives occur when a test is Disabled, Skipped, or Filtered

Report All Un-Executed Assertions

- On test exit, run through all the identified assertions and report any not Executed

Prototype Implementation (in LLVM, not upstream `googletest`)

Builds and Runs with Clang/Linux, gcc/Linux, and MSVC/Windows

- I don’t have access to a Mac, so haven’t tried it on macOS
- Each combo has its own weird way of getting the static allocation to work

Has Found Real Test Issues

- Fixed some test bugs, mostly zero-trip loops and unreachable cases
- Test author refactored one test
- Another test was missing initialization in some methods

Still Too Many False Positives

- Failed assertions not flagged as Executed – this is a quirk of how the `EXPECT/ASSERT` macros work
- Does not integrate smoothly with `googlemock`
- Latest import has a new Skip feature, doesn’t play well with current Rotten detection

Wins!

Hash	Review	PR	Description
63f9505	D95255		[ADT] Remove test assertion that will not be executed
6ea7ecb	D95256		Don’t use EXPECT* macros in a subprocess that exits by signalling
25fefa5	D95259		[TextAPI] Remove a zero-trip loop and the assertions within it
a0749f9	D95258		[ProfileData] Correct a test assertion
98754e2	D95257		[GlobalISel] Add missing setUp() calls to legalizer unit tests
05eeb60	D98518		RPCUtilsTest.cpp, replace un-executed EXPECT with unreachable
fb4f605			Recode more unreachable assertions and tautologies
b7578f9			Tweak test so assertion is always executed
206343f			Disable some tests on Windows at compile-time, not runtime
2b72954		PR49273	ConstantRangeTest.cpp missing things in “exhaustive” test
		PR49557	StencilTest.cpp has an EXPECT_THAT never executed
		PR49558	TokensTest.cpp has an EXPECT_THAT never executed
		PR49561	OpenMPIRBuilderTest.cpp is missing cases in 3 places
		PR49562	QualTypeNamesTest does nothing

Plans Going Forward

Contribute to Open-Source `googletest` Project

- Will get better design/code review from the project experts
- Using `googletest`'s own test suite will help work out how to better integrate with other features
- Have received internal Sony approval to make this contribution

Import to LLVM as a Usual Update from Upstream

- Ensures consistency with upstream implementation, limits local patches on the LLVM side

Go Back to Fixing Clang/LLVM Unit Tests

- Improves the quality of our Unit Tests using known technology

Think About Applying This to Other Kinds of Tests

- `lit` tests (basically, `FileCheck`) easy to mess up

Thanks for listening!

Sony
Interactive
Entertainment