Uncategorized

Unit Testing + Coverage: A Power Duo for Clean Code!

FEBRUARY 24, 2025

Nishma N J

Developer thinks???

Unit test setup

I am here to help!!!

Does my code really work as expected???

...

Let’s find out! I’ll test each function and method in isolation to ensure they behave correctly.

...

Hey, Who are you???

I’m your code’s personal **bodyguard**! 🛡️ I test each function and method in isolation to ensure they work **perfectly** before you integrate them.

....

...

...

...

So, what exactly do you do?

I focus on the **smallest testable parts** of your application—like functions and methods—making sure they behave **just the way you expect**.

...

...

...

And your ultimate goal?

Simple! I validate that every unit of your software is functioning **correctly** so you can build **reliable, bug-free applications** with confidence!

...

...

...

...

...

Sounds great! How do I start???

Easy! Follow these steps:”

- Write test cases for individual functions.
- Use frameworks like **Unity, Ceedling, or GoogleTest** (for C/C++).
- Run tests and validate expected vs. actual results.

...

...

...

...

...

Getting Started with Unit Testing !!!

Unit Testing

Unit testing is the process of testing individual functions, modules, or components of the software in isolation to verify that each unit behaves as expected.

Steps to Start Unit Testing

1. **Choose a Unit Testing Framework**
   - For **Embedded C** projects, popular choices include:
     - **Ceedling** (Unity + CMock + CException)
     - **Unity** (Lightweight, single-file framework)
     - **CppUTest** (For C and C++ unit testing)
2. **Set Up the Environment**
   - Install the selected framework (e.g., `ceedling new project` for Ceedling).
   - Configure the **Makefile / CMake** to include the test framework.
   - Create a separate `test/` folder to store unit test files.
3. **Write Test Cases**
   - Follow the **AAA (Arrange-Act-Assert) pattern**:
     - **Arrange**: Set up the test environment, initialize variables.
     - **Act**: Call the function under test.
     - **Assert**: Verify expected vs. actual output.

```
#include "unity.h"
#include "my_module.h"

void test_Addition(void) {
    TEST_ASSERT_EQUAL(5, add(2, 3));
}

void setUp(void) {}  // Runs before each test
void tearDown(void) {}  // Runs after each test

int main(void) {
    UNITY_BEGIN();
    RUN_TEST(test_Addition);
    return UNITY_END();
}
```

4. **Run the Tests**
   - Compile and execute using `make test` or `ceedling test:all`.

Hey, My tests runs fine!!!

....

....

....

....

....

....

....

Hey wait.....

How will I apply it to SonarQube???

Don't worry...

Here is my friend SonarQube to help you further...

..

.

.

.

.

.

.

.

Hey... Meet him

Hey SonarQube, I’ve got my unit tests ready. Can you help me with coverage reports?

Hi, I am SonarQube.

How can I help you???

Oh, you mean using `gcov`, `lcov`, or `GCC’s --coverage` flag?

Of course! But first, did you run your tests with coverage enabled?

Hey SonarQube,

If I want to generate a coverage report myself using my unit test results—for example, while using Code::Blocks—how should I get started?

Exactly! I generate a **.gcda** or **.xml** file containing coverage data while running your tests.

Great! Now, feed me that coverage report. Just use:

```
sonar-scanner -
Dsonar.coverageReportPaths=coverage.xm
l
```

I'll analyze it and show you how much of your code is actually tested!

.

.

..

...

Done! What's next?

..

..

..

Okay.....

..

First, let's ensure your code is **buildable**. Run this command:

```
cb_console_runner.exe --build
Unit_Test_sonarQube_coverage
```

..

.

.

.

.

.

...

The build is complete.
What's next?

Now, we need to generate a **JSON file** for analysis. Use these two steps:

```
i.build-wrapper-win-x86-64.exe --out-dir
bw-output codeblocks.exe --clean
Unit_Test_sonarQube_coverage.cbp
ii.build-wrapper-win-x86-64.exe --out-dir
bw-output codeblocks.exe --rebuild
Unit_Test_sonarQube_coverage.cbp
```

Great! Now, let's build the project properly. Go ahead with:

```
mkdir build && cd build
cmake -G "MinGW Makefiles" ..
mingw32-make
```

Awesome! Now I can check my test coverage and spot untested code in the **SonarQube dashboard**!

Now, generate the **coverage report** by running this in the root folder

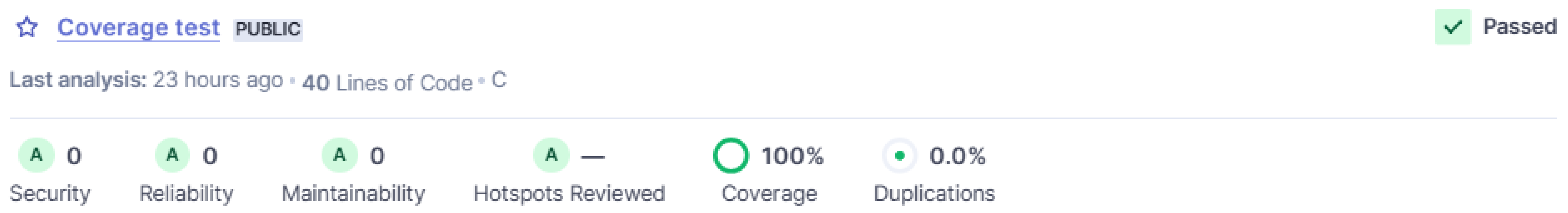
```
cd ..
gcovr -r . --sonarqube -o coverage.xml
```

Wow! So now, I can easily spot **untested code** and improve my test coverage?

Perfect! Now let me analyze it. Run

```
sonar-scanner -X
```

Exactly! More coverage = **better quality code**! Keep testing and refining



Coverage test PUBLIC

Passed

Last analysis: 23 hours ago • 40 Lines of Code • C

A 0	A 0	A 0	A —	100%	0.0%
Security	Reliability	Maintainability	Hotspots Reviewed	Coverage	Duplications

#Technical

Nishma N J

Friendly natured. Born and brought up at "scotland of India" (Coorg) Basically a singer. Also interested in pencil sketches.

View all posts by Nishma N J