Coding Style / Test-driven development

임종우 (Jongwoo Lim)

C++ Style Guide

Refer http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

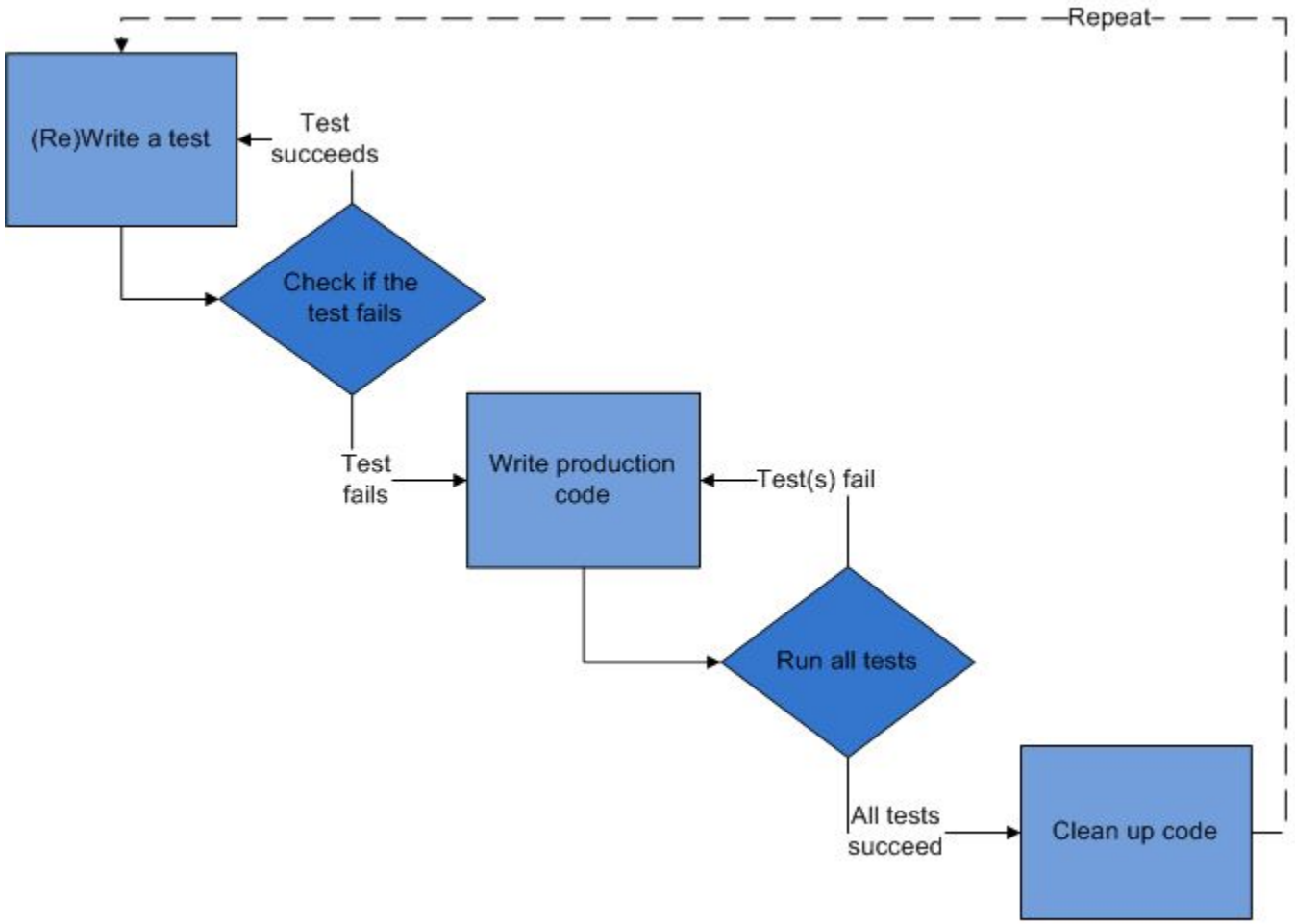
Test-Driven Development

- Repetition of a very short development cycle:
  - First the developer writes an (initially failing) automated test case that defines a desired improvement or new function.
  - Then produces the minimum amount of code to pass that test, and
  - Finally refactors the new code to acceptable standards.

- TDD encourages simple designs and inspires confidence.

- "You aren't gonna need it" (YAGNI).
  - By focusing on writing only the code necessary to pass tests, designs can often be cleaner and clearer than is achieved by other methods.

Test-Driven Development



[Wikipedia]

Test-Driven Development

- Separate common set up and teardown logic into test support services.
- Keep each test oracle focused on only the results necessary to validate its test.
- Treat your test code with the same respect as your production code. It also must work correctly for both positive and negative cases, last a long time, and be readable and maintainable.
- Get together with your team and review your tests and test practices to share effective techniques and catch bad habits. It may be helpful to review this section during your discussion.

Test-Driven Development

```
#include <iostream>
#include <glog/logging.h>
#include <gtest/gtest.h>
#include "image_file.h"

using namespace std;
using namespace rvslam;

namespace {

TEST(ImageFileTest, TestReadImageGray8) {
  MCImageGray8 mc_gray;
  EXPECT_TRUE(ReadImage("test_gray8.png", &mc_gray));
  LOG(INFO) << "mc_gray " << mc_gray.rows() << "x" << mc_gray.cols();
  EXPECT_TRUE(WriteImageGray8(mc_gray, "test_mc_gray8_out.png"));
  EXPECT_TRUE(ReadImage("test_rgb8.png", &mc_gray));
  EXPECT_TRUE(WriteImageGray8(mc_gray, "test_rgb_mc_gray8_out.png"));
}

}  // namespace

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Test-Driven Development

```
...

TEST(BinarySearchTest, BinarySearchTest) {
  const int sorted_array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  EXPECT_EQ(2, BinarySearch(sorted_array, 3));
  EXPECT_EQ(8, BinarySearch(sorted_array, 9));
  EXPECT_EQ(-1, BinarySearch(sorted_array, 0));
  EXPECT_EQ(-1, BinarySearch(sorted_array, 100));
  EXPECT_EQ(-1, BinarySearch(NULL, 100));
}

TEST(SortTest, SortTest) {
  int array[] = { 9, 2, 1, 8, 5, 7, 10, 3, 4, 6 };
  EXPECT_FALSE(Sort(NULL, 3));
  EXPECT_TRUE(Sort(array, 5));
  for (int i = 1; i < 5; ++i) EXPECT_LE(array[i - 1], array[i]);
  EXPECT_TRUE(Sort(array, 10));
  const int sorted_array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  for (int i = 0; i < 10; ++i) EXPECT_EQ(array[i], sorted_array[i]);
}

...
```

Test-Driven Development

Practices to avoid:

- Do not have test cases depend on system state manipulated from previously executed test cases.
- A test suite where test cases are dependent upon each other is brittle and complex.
- Interdependent tests can cause cascading false negatives.
- Do not test precise execution behavior timing or performance.
- Do not try to build “all-knowing oracles.” An oracle that inspects more than necessary is more expensive and brittle over time than it needs to be.
- Testing implementation detail
- Slow tests

Test-Driven Development

- Unit test vs. Functional test
  - Unit test : testing if a function (in a class) is working as designed.
  - Functional test : testing some functionality in the system.

- Write simple and fast tests.
  - Sometimes you need to ‘simulate’ some of devices, such as database, network communication, etc.

Test-Driven Development

- Test Double : test-specific capability that substitutes for a system capability
  - Dummy – A function returning a default value where required.
  - Stub – A stub adds simplistic logic to a dummy, providing different outputs.
  - Mock – A mock is specified by an individual test case to validate test-specific behavior, checking parameter values and call sequencing.
  - Simulator – A simulator is a comprehensive component providing a higher-fidelity approximation of the target capability (the thing being doubled). A simulator typically requires significant additional development effort.