TEST-DRIVEN DEVELOPMENT IN C++

Richard Thomson
Senior Software Engineer
Fusion-io
@LegalizeAdulthd
http://LegalizeAdulthood.wordpress.com
legalize@xmission.com

Outline

- ~60-90 minutes: Walkthrough TDD
  - Designed to give you exposure to TDD in a worked example.
  - You follow allong at your computer, step by step.
- ~80-110 minutes: Self study exercise
  - Designed to give you exposure to TDD as a design activity.
  - Pair programming encouraged!
- ~10 minutes: wrap-up discussion

USB Drive Material

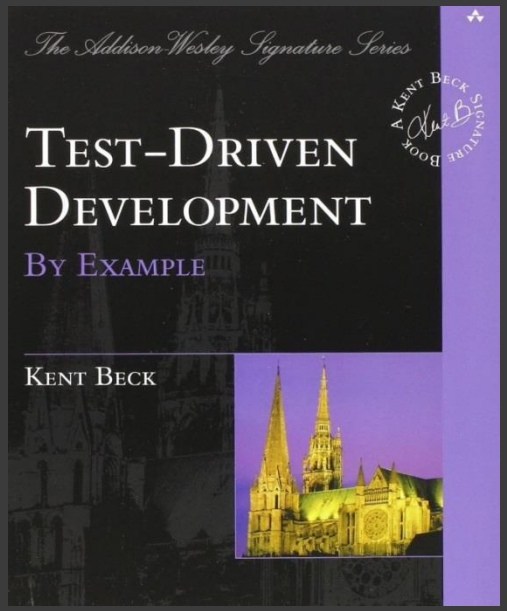
- CMake 2.8
- Boost 1.55
- Turtle 1.2.5 (includes docs)
- Boost.Test documentation rewrite
- Step-by-step code folders

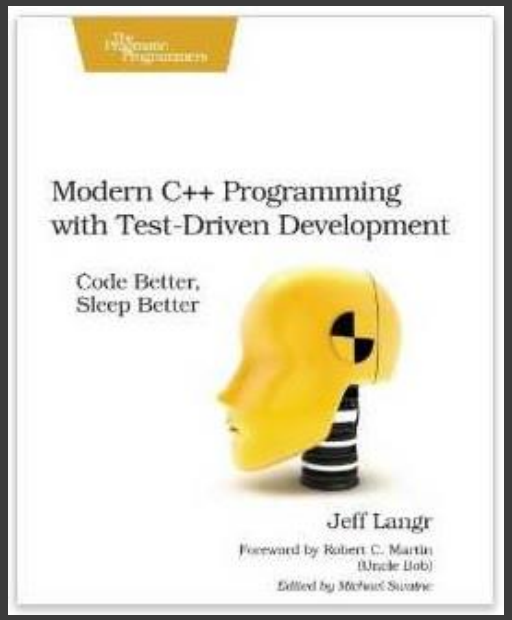
Getting Ready

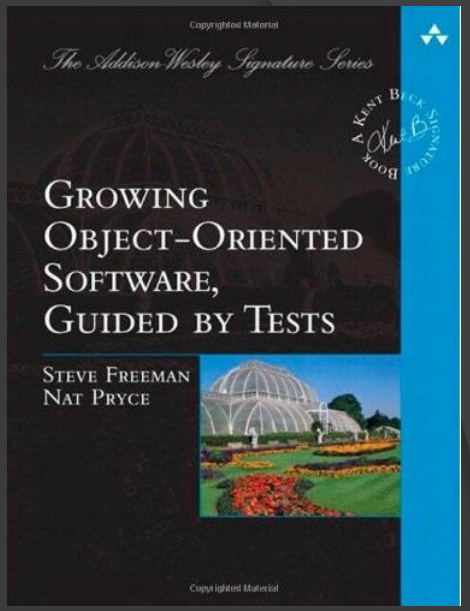
- WiFi:
- Check your C++ compiler
- Download and install Boost ≥ 1.55
  - http://www.boost.org
  - Boost.Test documentation: http://user.xmission.com/~legalize/boost.test
- Download and install Turtle
  - http://turtle.sourceforge.net

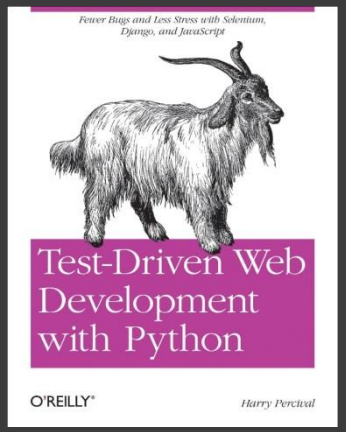
1. Only write production code to make a test pass.
2. Only write as much of a test as needed to make it fail; compilation failures are failures.
3. Only write just enough production code to make a test pass.
4. Refactor only when tests are passing.

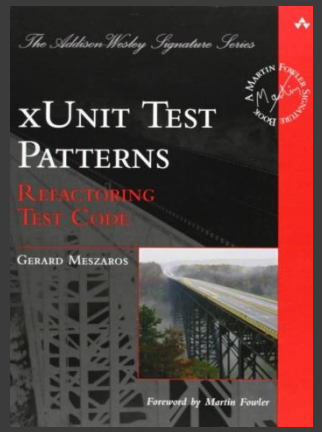
Rules of Test-Driven Development











This is only an introduction...

Prime Factors

Write a function named `prime_factors` that:

- takes an integer, *n*
- returns a `std::vector<int>` containing all the prime factors of *n* in numerical order
- 1 is not a prime factor.

- Create a static library factors for your implementation
- Create a console executable test_factors for your tests
- Successful build runs test_factors
- Non-zero exit code of test_factors fails build

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

CMake version required for this project.

1st Test: 1 → { }


```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

CMake C++ project named prime_factors

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

Adds a static library named factors to the project. The library is built from factors.cpp and factors.h.

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

Include a file that contains local paths specific to your system.

1st Test: 1 → { }


```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
```

1st Test: 1 → { }


```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
```

Set a variable to tell find_package where it can find Boost.

1st Test: 1 → { }

```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
```

This path is specific to your system.

You must use slashes (/), even on Windows.

1st Test: 1 → { }

Configure some Boost settings and use find_package to get Boost 1.55

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/f
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

1st Test: 1 → { }


```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

Add an executable test_factors to the project. It is built from test_factors.cpp.

1st Test: 1 → { }


```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

Test code depends on production code. test_factors depends on factors.

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/f
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

test_factors needs Boost include directory in its include search path.

1st Test: 1 → { }


```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

test_factors has a link dependency on factors

1st Test: 1 → { }


```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)
```

Run the tests on every successful build.

1st Test: 1 → { }

Windows:

VS 2010: `cmake -G "Visual Studio 10"`

VS 2012: `cmake -G "Visual Studio 11"`

VS 2013: `cmake -G "Visual Studio 12"`

Unix:

```
cmake -G "Unix Makefiles"
```

Macintosh:

```
cmake -G Xcode
cmake -G "Unix Makefiles"
```

To see a list of supported generators on your system:

```
cmake --help
```

1st Test: 1 → { }

```
// CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(prime_factors CXX)
add_library(factors STATIC
  factors/factors.cpp factors/factors.h)
include(LocalPaths.txt)
set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.55)
add_executable(test_factors
  test/test_factors.cpp)
add_dependencies(test_factors factors)
target_include_directories(test_factors
  PRIVATE . ${Boost_INCLUDE_DIRS})
target_link_libraries(test_factors factors)
add_custom_command(TARGET test_factors
  POST_BUILD COMMAND test_factors)

unresolved external symbol _main
```

1st Test: 1 → { }


```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
```

1st Test: 1 → { }


```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
```

Instructs Boost.Test to provide a definition for `main()`

1st Test: 1 → { }


```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
```

Header-only version of Boost.Test; simpler build configuration but longer compile times

1st Test: 1 → { }

```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>

EXEC : Test setup error : test tree is empty
```

1st Test: 1 → { }


```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
#include "factors.h"
```

1st Test: 1 → { }


```
// test_factors.cpp
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
#include "factors.h"

'factors.h': No such file or directory
```

1st Test: 1 → { }

```
// CMakeLists.txt
target_include_directories(test_factors
    PRIVATE factors ${Boost_INCLUDE_DIRS})
```

1st Test: 1 → { }


```
// CMakeLists.txt
target_include_directories(test_factors
    PRIVATE factors ${Boost_INCLUDE_DIRS})
```

Add the factors directory to the include search path for test_factors

1st Test: 1 → { }

```
// CMakeLists.txt
target_include_directories(test_factors
    PRIVATE factors ${Boost_INCLUDE_DIRS})
```

```
EXEC : Test setup error : test tree is empty
```

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yi
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Declares the next block of code as the named, automatically registered test case. The test case is an instance of a class and the block of code defines the test method.

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

The name of the test case; a descriptive phrase for the feature being tested as a C++ identifier.

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

The expected factors for 1 -- an empty vector

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

The system under test.

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yie
{
  vector<int> expected;

  vector<int> actual = prime

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

An assertion macro that compares two collections via their forward iterators. The assertion fails when the collections don't match in size or in content. A failed assertion fails the test case.

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yie
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Anatomy of a test case:

Phase 1: Setup

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yie
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Anatomy of a test case:

Phase 2: Execute

1st Test: 1 → { }


```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yie
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Anatomy of a test case:

Phase 3: Verify

1st Test: 1 → { }

```
// test_factors.cpp
using namespace std;
BOOST_AUTO_TEST_CASE(one_yie
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Anatomy of a test case:

Separate Phases Visually

1st Test: 1 → { }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
'prime_factors': identifier not found
```

1st Test: 1 → { }


```
// factors.h
#if !defined(FACTORS_H)
#define FACTORS_H
#include <vector>
extern std::vector<int>
  prime_factors(int n);
#endif
```

1st Test: 1 → { }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
unresolved external symbol
"std::vector<int> prime_factors(int)"
```

1st Test: 1 → { }

```
// factors.cpp
#include "factors.h"
#include <stdexcept>
extern std::vector<int>
prime_factors(int n)
{
  throw std::runtime_error(
    "not implemented");
}
```

1st Test: 1 → { }

```
// factors.cpp
#include "factors.h"
#include <stdexcept>
extern std::vector<int>
prime_factors(int n)
{
  throw std::runtime_error(
    "not implemented");
}
```

Force a failure to ensure that this code is called.

1st Test: 1 → { }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
fatal error in "one_yields_empty":
  std::runtime_error: not implemented
```

1st Test: 1 → { }

```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COL
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Boost.Test treats unhandled exceptions as failures and prints information from the exception

```
fatal error in "one_yields_empty":
  std::runtime_error: not implemented
```

1st Test: 1 → { }

```
// factors.cpp
#include "factors.h"
#include <stdexcept>

extern std::vector<int>
prime_factors(int n)
{
  return std::vector<int>();
}
```

1st Test: 1 → { }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  vector<int> expected;

  vector<int> actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
*** No errors detected
```

1st Test: 1 → { }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(two_yields_2)
{
  vector<int> expected;
  expected.push_back(2);

  vector<int> actual = prime_factors(2);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

2nd Test: 2 → { 2 }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(two_yields_2)
{
  vector<int> expected;
  expected.push_back(2);

  vector<int> actual = prime_factors(2);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
```

Expected factors: 2

2nd Test: 2 → { 2 }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(two_yields_2)
{
  vector<int> expected;
  expected.push_back(2);

  vector<int> actual = prime_factors(2);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
fatal error in "two_yields_2": critical check
{ begin(expected), end(expected) } ==
{ begin(actual), end(actual) } failed.
Collections size mismatch: 1 != 0
```

2nd Test: 2 → { 2 }

```
// factors.cpp
extern std::vector<int>
prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    primes.push_back(2);
  }
  return primes;
}
```

2nd Test: 2 → { 2 }

```
// factors.cpp
extern std::vector<int>
prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    primes.push_back(2);
  }
  return primes;
}
```

We're doing the smallest change we can to pass the test

2nd Test: 2 → { 2 }

```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(two_yields_2)
{
  vector<int> expected;
  expected.push_back(2);

  vector<int> actual = prime_factors(2);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
*** No errors detected
```

2nd Test: 2 → { 2 }


```
// test_factors.cpp
BOOST_AUTO_TEST_CASE(two_yields_2)
{
  vector<int> expected;
  expected.push_back(2);

  vector<int> actual = prime_factors(2);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(expected), end(expected),
    begin(actual), end(actual));
}
*** No errors detected
```

Eliminate Duplication

2nd Test: 2 → { 2 }


```
// test_factors.cpp
struct fixture {
  vector<int> expected;
  vector<int> actual;
};

BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  fixture f;

  f.actual = prime_factors(1);

  BOOST_REQUIRE_EQUAL_COLLECTIONS(
    begin(f.expected), end(f.expected),
    begin(f.actual), end(f.actual));
}
*** No errors detected
```

Extract State in Struct


```
// test_factors.cpp
struct fixture {   // ...
  void prime_factors(int n) {
    actual = ::prime_factors(n);
  }
  void verify_expected_factors() {
    BOOST_REQUIRE_EQUAL_COLLECTIONS(
      begin(expected), end(expected),
      begin(actual), end(actual));
  }
};
BOOST_AUTO_TEST_CASE(one_yields_empty)
{
  fixture f;

  f.prime_factors(1);

  f.verify_expected_factors();
}
*** No errors detected
```

Extract Methods

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(one_yields_empty, fixture)
{
  prime_factors(1);

  verify_expected_factors();
}
*** No errors detected
```

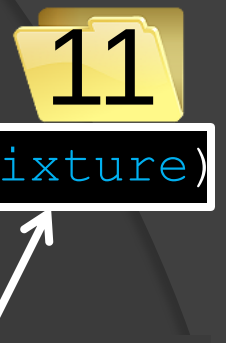
Use Fixture


```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(one_yields_empty, fixture)
{
  prime_factors(1);

  verify_expected_factor
}
*** No errors detected
```

Declares an automatically registered test case with access to variables and methods of a fixture.

Use Fixture

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(one_yields_empty, fixture)
{
  prime_factors(1);

  verify_expected_factor
}
*** No errors detected
```



The name of the fixture class that will serve as the base class for the test case class.

Fixtures allow you to eliminate duplication between test cases.

Use Fixture


```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(two_yields_2, fixture)
{
  expected.push_back(2);

  prime_factors(2);

  verify_expected_factors();
}
*** No errors detected
```

Use Fixture


```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(three_yields_3, fixture)
{
  expected.push_back(3);

  prime_factors(3);

  verify_expected_factors();
}
```

3rd Test: 3 → { 3 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(three_yields_3, fixture)
{
  expected.push_back(3);

  prime_factors(3);

  verify_expected_factors();
}
fatal error in "three_yields_3": critical check
{ begin(expected), end(expected) } ==
{ begin(actual), end(actual) } failed.
Mismatch in a position 0: 3 != 2
```

3rd Test: 3 → { 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
      primes.push_back(n);
  }
  return primes;
}
*** No errors detected
```

3rd Test: 3 → { 3 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(four_yields_2_2, fixture)
{
  expected.push_back(2);
  expected.push_back(2);

  prime_factors(4);

  verify_expected_factors();
}
```

4th Test: 4 → { 2, 2 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(four_yields_2_2, fixture)
{
  expected.push_back(2);
  expected.push_back(2);

  prime_factors(4);

  verify_expected_factors();
}
fatal error in "four_yields_2_2": critical check
{ begin(expected), end(expected) } ==
{ begin(actual), end(actual) } failed.
Mismatch in a position 0: 2 != 4
Collections size mismatch: 2 != 1
```

4th Test: 4 → { 2, 2 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    if (n % 2 == 0) {
      primes.push_back(2);
      n /= 2;
    }
    if (n > 1) {
      primes.push_back(n);
    }
  }
  return primes;
}
*** No errors detected
```

4th Test: 4 → { 2, 2 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(six_yields_2_3, fixture)
{
  expected.push_back(2);
  expected.push_back(3);

  prime_factors(6);

  verify_expected_factors();
}
```

5th Test: 6 → { 2, 3 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(six_yields_2_3, fixture)
{
  expected.push_back(2);
  expected.push_back(3);

  prime_factors(6);

  verify_expected_factors();
}
```

```
*** No errors detected
```

A test that passes unexpectedly is cause for careful examination.

Did we make a mistake in our test case?

Did this test case fail to push our design of the system to the next level?

5th Test: 6 → { 2, 3 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(
    eight_yields_2_2_2, fixture)
{
  expected.push_back(2);
  expected.push_back(2);
  expected.push_back(2);

  prime_factors(8);

  verify_expected_factors();
}
```

6th Test: 8 → { 2, 2, 2 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(eight_yields_2_2_2, fixture)
{
  expected.push_back(2);
  expected.push_back(2);
  expected.push_back(2);

  prime_factors(8);

  verify_expected_factors();
}
fatal error in "eight_yields_2_2_2": critical check
{ begin(expected), end(expected) } ==
{ begin(actual), end(actual) } failed.
Mismatch in a position 1: 2 != 4
Collections size mismatch: 3 != 2
```

6th Test: 8 → { 2, 2, 2 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    while (n % 2 == 0) {        // !!!
      primes.push_back(2);
      n /= 2;
    }
    if (n > 1) {
      primes.push_back(n);
    }
  }
  return primes;
}
*** No errors detected
```

6th Test: 8 → { 2, 2, 2 }

```
// test_factors.cpp
struct fixture {
  vector<int> expected;
  vector<int> actual;
  void prime_factors(int n) {
    actual = ::prime_factors(n);
  }
  ~fixture() {
    BOOST_REQUIRE_EQUAL_COLLECTIONS(
      expected.begin(), expected.end(),
      actual.begin(), actual.end());
  }
};
```

Validate in Destructor


```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(one_yields_empty, fixture)
{
  prime_factors(1);

  verify_expected_factors();
}

BOOST_FIXTURE_TEST_CASE(two_yields_2, fixture)
{
  expected.push_back(2);

  prime_factors(2);

  verify_expected_factors();
}

// remove verify_expected_factors in other cases...
```

Validate in Destructor

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(one_yields_empty, fixture)
{
  prime_factors(1);

  verify_expected_factors();
}

BOOST_FIXTURE_TEST_CASE(two_yields_2, fixture)
{
  expected.push_back(2);

  prime_factors(2);

  verify_expected_factors();
}
```

You may prefer a little repetition over implicit verification. Always keep your tests readable.

...or not

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(nine_yields_3_3, fixture)
{
  expected.push_back(3);
  expected.push_back(3);

  prime_factors(9);
}
```

7th Test: 9 → { 3, 3 }

```
// test_factors.cpp
BOOST_FIXTURE_TEST_CASE(nine_yields_3_3, fixture)
{
  expected.push_back(3);
  expected.push_back(3);

  prime_factors(9);
}
fatal error in "nine_yields_3_3": critical check
{ begin(expected), end(expected) } ==
{ begin(actual), end(actual) } failed.
Mismatch in a position 0: 3 != 9
Collections size mismatch: 2 != 1
```

7th Test: 9 → { 3, 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    int candidate = 2;
    while (n % candidate == 0) {
      primes.push_back(candidate);
      n /= candidate;
    }
    if (n > 1) {
      primes.push_back(n);
    }
  }
  return primes;
}
```

7th Test: 9 → { 3, 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  if (n > 1) {
    int candidate = 2;
    while (n % candidate == 0) {
      primes.push_back(candidate);
      n /= candidate;
    }
  }
  if (n > 1) {
    primes.push_back(n);
  }
  return primes;
}
```

7th Test: 9 → { 3, 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  int candidate = 2;
  while (n > 1) {                    // !!!
    while (n % candidate == 0) {
      primes.push_back(candidate);
      n /= candidate;
    }
    ++candidate;
  }
  if (n > 1) {
    primes.push_back(n);
  }
  return primes;
}
*** No errors detected
```

7th Test: 9 → { 3, 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  int candidate = 2;
  while (n > 1) {
    while (n % candidate == 0) {
      primes.push_back(candidate);
      n /= candidate;
    }
    candidate++;
  }
  if (n > 1) {
    primes.push_back(n);
  }
  return primes;
}
*** No errors detected
```

7th Test: 9 → { 3, 3 }

```
// factors.cpp
extern std::vector<int> prime_factors(int n)
{
  std::vector<int> primes;
  int candidate = 2;
  while (n > 1) {
    for (; n % candidate == 0; n /= candidate) {
      primes.push_back(candidate);
    }
    candidate++;
  }
  return primes;
}
*** No errors detected
```

7th Test: 9 → { 3, 3 }

- Algorithm evolved with each test case
- Evolved from conditionals to loops
- Cleaned up the algorithm at the end
- Refactoring is a chance to improve design
- Passing tests are a safety net for refactoring
- TDD is about design as an activity
- Don't leave stinky code in place for long
- Design your build along with your code

Ah-ha!

User Interface

- Provide a dialog where the user is prompted for the value that will be passed to `prime_factors`.
- The dialog contains a text box and an OK button.
- Initially the OK button is disabled.
- As the user types each character in the text box, the OK button will be enabled as soon as the entered text is a valid integer.

Humble Dialogs and Mediators

- Use the "Humble Dialog" design pattern
- Put all the behavior in a Mediator class
- Drive the mediator/dialog design with tests
- You can test anything with the right design, ...even UI behavior

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

// CMakeLists.txt
add_executable(test_factors
    test/test_factors.cpp test/test_mediator.cpp)
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

// CMakeLists.txt
add_executable(test_
    test/test_factor
```

Instructs Boost.Test to inhibit any automatic linking. Any definitions needed for the test framework were provided by the header-only version of the library included in test_factors.cpp.

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

// CMakeLists.txt
add_executable(test_
    test/test_factor
```

The header for Boost.Test that provides declarations, but not definitions.

This is how we comply with the ODR when using the header-only version of Boost.Test and multiple compilation units.

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

// CMakeLists.txt
add_executable(test_factors
    test/test_factors.cpp test/test_mediator.cpp)
```

Header file for the system under test -- the mediator

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

// CMakeLists.txt
add_executable(test_factors
    test/test_factors.cpp test/test_mediator.cpp)
```

Add new test source file to the executable

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

'mediator.h': No such file or directory
```

Button initially disabled

```
// mediator.h
#if !defined(MEDIATOR_H)
#define MEDIATOR_H

#endif

// CMakeLists.txt
add_library(factors STATIC
    factors/factors.cpp factors/factors.h
    factors/mediator.h)
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  // ?what_type? dialog;

  prime_factors_mediator mediator(dialog);

  // ?how? verify that ok button is initially disabled
}
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  // ?what_type? dialog;

  prime_factors_mediator mediator(dialog);

  // ?how? verify that ok button is initially disabled
}
'prime_factors_mediator' : undeclared identifier
'dialog' : undeclared identifier
```

Button initially disabled

```
// mediator.h
class prime_factors_dialog
{
public:
  virtual ~prime_factors_dialog() { }
};

class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog) { }
};
```

Button initially disabled

```
// mediator.h
class prime_factors_dialog
{
public:
  virtual ~prime_factors_dial
};

class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog) { }
};
```

Abstract interface for some dialog that does not yet exist in our system.

Button initially disabled

```
// mediator.h
class prime_factors_dialog
{
public:
  virtual ~prime_factors_dialog() { }
};

class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog) { }
};
```

The system under test we are driving to interact with dialog.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
```

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initi
}
```

Meh. This sucks, but it gets us past a compile error. We know we're going to change it soon, but we focus on making progress right now.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_butto                    )
{
  prime_factors_dialog dialog

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
```

This assertion fails the test if it's argument evaluates to false.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_butto                    )
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
```

This gives us a useful message for what we really want to assert and yields false to make the test fail.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
fatal error in "ok_button_initially_disabled":
critical check !"verify that ok button is initially
disabled" failed
```

Button initially disabled

```
// mediator.h
class prime_factors_dialog
{
public:
  virtual ~prime_factors_dialog() { }
  virtual void enable_ok_button(bool enabled) = 0;
};
```

Button initially disabled

```
// mediator.h
class prime_factors_dialog
{
public:
  virtual ~prime_factors_dialog() { }
  virtual void enable_ok_button(bool enabled) = 0;
};
```

We need some way to tell the dialog to enable the OK button, so we pull interface methods into existence as we need them.

This is incremental design of collaborators, driven by tests.

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
'prime_factors_dialog' : cannot instantiate abstract class
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#inc
BOOS
{
  pr
  pr
  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
```

We need some way to create a standin for this dialog so that we can verify the interactions between the mediator and the dialog.

We could write a test double by hand, but it gets quite tedious very quickly.

Let's use a mock provided by Turtle instead.

```
'prime_factors_dialog' : cannot instantiate abstract class
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
```

Button initially disabled


```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
'turtle/mock.hpp': No such file or directory
```

Button initially disabled

```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
set(TURTLE_INCLUDE D:/Code/turtle)

// CMakeLists.txt
target_include_directories(test_factors
  PRIVATE factors ${TURTLE_INCLUDE} ${Boost_INCLUDE_DIRS})
```

Button initially disabled

```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
set(TURTLE_INCLUDE D:/Code/turtle)

// CMakeLists.txt
target_include_directories(test_factors
  PRIVATE factors ${TURTLE_INCLUDE} ${Boost_INCLUDE_DIRS})
```

Set a CMake variable to the location of turtle.

This location is specific to your system.

Use slashes (/), even on Windows.

Button initially disabled

```
// LocalPaths.txt
set(BOOST_INCLUDEDIR D:/Code/boost/boost_1_55_0)
set(TURTLE_INCLUDE D:/Code/turtle)

// CMakeLists.txt
target_include_directories(test_factors
  PRIVATE factors ${TURTLE_INCLUDE} ${Boost_INCLUDE_DIRS})
```

Include Turtle in the include search path. Turtle is a header-only library and needs no link changes.

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  prime_factors_dialog dialog;

  prime_factors_mediator mediator(dialog);

  BOOST_REQUIRE(
    !"verify that ok button is initially disabled");
}
'prime_factors_dialog' : cannot instantiate abstract class
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Declares a mock class that derives from a base class containing virtual methods.

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

The name of the mock class

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

The name of the class to be mocked

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Declares a mock for a method

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

The name of the mocked method

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1)
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Number of arguments

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Instantiate the mock

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Expect a mock call

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Expected method

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Expected call count

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Expected arguments

Button initially disabled

```
// test_mediator.cpp
#define BOOST_TEST_NO_LIB
#include <boost/test/unit_test.hpp>
#include <turtle/mock.hpp>
#include "mediator.h"

MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  MOCK_METHOD(enable_ok_button, 1);
};

BOOST_AUTO_TEST_CASE(ok_button_initially_
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
```

Expectations verified automatically upon destruction of the mock

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);

  prime_factors_mediator mediator(dialog);
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
. once().with( false )
```

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_o                    );

  prime_factors_mediator mediator(dialog);
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
. once().with( false )
```

We told Turtle something should happen and it didn't.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_o                    );

  prime_factors_mediator mediator(dialog);
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
. once().with( false )
```

Turtle knows that dialog is an instance of mock_dialog.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_o                    );

  prime_factors_mediator mediator(dialog);
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
. once().with( false )
```

The method that was expected to be called, but wasn't.

Button initially disabled

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_initially_disabled)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_o                    );

  prime_factors_mediator mediator(dialog);
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
. once().with( false )
```

The call count and parameters that we expected, but didn't get.

Button initially disabled

```
// mediator.h
class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog) {
    dialog.enable_ok_button(false);
  }
};
*** No errors detected
```

Button initially disabled

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Add a way to get the value text

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_                    er)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

A mock must meet all its expectations. Unless specified, the expectations can be met in any order.

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Specifies a return value when the expected arguments match.

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CL
{
  // ...
  MOCK_METHO
};
BOOST_AUTO_T                                    nteger)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

With no call count specified, an expected method can be called zero or more times.

Be careful not to overspecify mock interactions.

In general, we allow queries any number of times and specify the cardinality of commands.

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Add a way to tell the mediator that the text value has changed.

Integer enables button

```
// test_mediator.cpp
MOCK_BASE_CLASS(mock_dialog, prime_factors_dialog)
{
  // ...
  MOCK_METHOD(value_text, 0);
};
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
'value_text' : is not a member of 'prime_factors_dialog'
'value_changed' : is not a member of 'prime_factors_mediator'
```

Integer enables button

```
// mediator.h
#include <string>

class prime_factors_dialog
{
public:
  virtual ~prime_factors_dialog() { }
  virtual void enable_ok_button(bool enabled) = 0;
  virtual std::string value_text() const = 0;
};
```

Integer enables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
'value_changed' : is not a member of
'prime_factors_mediator'
```

Integer enables button

```
// mediator.h
class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog* dialog) {
    dialog->enable_ok_button(false);
  }
  void value_changed() { }
};
```

Integer enables button


```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
untriggered expectation:
dialog.mock_dialog::enable_ok_button
```

Integer enables button

```
// mediator.h
class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog)
      : dialog_(dialog) {
    dialog.enable_ok_button(false);
  }
  void value_changed() {
    dialog_.enable_ok_button(true);
  }

private:
  prime_factors_dialog& dialog_;
};
```

Integer enables button


```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
*** No errors detected
```

Integer enables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

What the hell?

* * *

We never queried the value for the text and we didn't even check that it was an integer, we just said "yeah, set the button to enabled"?!?!?

Integer enables button


```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(ok_button_enabled_with_valid_integer)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button).once().with(false);
  MOCK_EXPECT(dialog.enable_ok_button).once().with(true);
  MOCK_EXPECT(dialog.value_text).returns("123");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Yes!

* * *

We are doing the minimal change to make the test pass. We know we have more work to do and that means we haven't yet written enough tests to tease out all the behavior of the system.

Integer enables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");

}
```

We should disable the button at least once, but we allow it to be disabled multiple times.
Don't overconstrain your collaborators.

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
unexpected call:
dialog.mock_dialog::enable_ok_button(true)
. at_least( 1/1 ).with( false )
```

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");
  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

This interaction was unexpected.

```
unexpected call:
dialog.mock_dialog::enable_ok_button(true)
. at_least( 1/1 ).with( false )
```

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");

                                          .alog);
}
unexpected call:
dialog.mock_dialog::enable_ok_button(true)
. at_least( 1/1 ).with( false )
```

We expected at least one call to this method with false and we got one call.

Empty text disables button

```
// mediator.h
class prime_factors_mediator
{
public:
  prime_factors_mediator(prime_factors_dialog& dialog)
      : dialog_(dialog) {
    dialog.enable_ok_button(false);
  }
  void value_changed() {
    dialog_.enable_ok_button(
      dialog_.value_text().length() > 0);
  }

private:
  prime_factors_dialog& dialog_;
};
```

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(empty_text_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
*** No errors detected
```

Empty text disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(not_a_number_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("junk");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
```

Not-a-Num disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(not_a_number_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("junk");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
unexpected call:
dialog.mock_dialog::enable_ok_button( true )
. at_least( 1/1 ).with( false )
```

Not-a-Num disables button

```
// mediator.h
#include <sstream>
class prime_factors_mediator
{
public:
  // ...
  void value_changed() {
    std::istringstream stream(dialog_.value_text());
    int n;
    stream >> n;
    dialog_.enable_ok_button(!stream.fail() &&
      dialog_.value_text().length() > 0);
  }
  // ...
};
```

Not-a-Num disables button

```
// test_mediator.cpp
BOOST_AUTO_TEST_CASE(not_a_number_disables_button)
{
  mock_dialog dialog;
  MOCK_EXPECT(dialog.enable_ok_button)
    .at_least(1).with(false);
  MOCK_EXPECT(dialog.value_text).returns("junk");

  prime_factors_mediator mediator(dialog);
  mediator.value_changed();
}
*** No errors detected
```

Not-a-Num disables button

Some Design Observations

- Defined dialog behavior without a dialog!
- Incrementally refined the behavior as we added tests
- Incrementally designed the abstract collaborator for the mediator
- Designed an interface by consuming it
- Behavior is not coupled to a specific framework (wxWidgets, Qt, MFC, etc.)

- Yes, at first.
- Learn to crawl before you walk.
- Learn to walk before you run.
- When you take a big step and it explodes, retreat to your last green test and retry in smaller steps.
- Commit every time you go green!

Are such tiny steps really necessary?

- We catch our "stupid mistakes" within seconds of making them, instead of minutes, hours, days, weeks or months later.
- We design our interfaces as consumers first, implementors second.
- The tests give us confidence to improve our design without introducing regressions.
- The tests eliminate fear of change!
- Studies have shown that TDD produces the same quality code in less time or higher quality code in the same amount of time.

Is all this test code worth it?

Are unit tests enough?

- No
- Unit Tests verify individual components
- Acceptance Tests verify integrated components as systems
- Manual acceptance tests are ok
- Automated acceptance tests are better
- FitNesse is an acceptance test framework

Exercise: Hangman Game

- Follow SOLID OOD principles http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- Test First!
- Rhythm: Red, Green, Refactor
- Concentrate on the collaboration between game and UI classes
- Don't worry about finishing the exercise, it is designed to consume more time than available