Is conditional compilation a valid mock/stub strategy for unit testing?

Asked 16 years, 3 months ago Modified 16 years, 3 months ago

Viewed 2k times

5

In a recent question on stubbing, many answers suggested C# interfaces or delegates for implementing stubs, but one answer suggested using conditional compilation, retaining static binding in the production code. This answer was modded -2 at the time of reading, so at least 2 people really thought this was a *wrong* answer. Perhaps misuse of DEBUG was the reason, or perhaps use of fixed value instead of more extensive validation. But I can't help wondering:

Is the use of conditional compilation an inappropriate technique for implementing unit test stubs? Sometimes? Always?

Thanks.

Edit-add: I'd like to add an example as a though experiment:

```
class Foo {
    public Foo() { .. }
    private DateTime Now {
      get {
#if UNITTEST_Foo
        return Stub_DateTime.Now;
#else
        return DateTime.Now;
#endif
      }
    }
    // .. rest of Foo members
}
```

comparing to

```
interface IDateTimeStrategy {
    DateTime Now { get; }
}
class ProductionDateTimeStrategy :
IDateTimeStrategy {
  public DateTime Now { get { return DateTime.Now;
} }
}
class Foo {
    public Foo() : Foo(new
ProductionDateTimeStrategy()) {}
    public Foo(IDateTimeStrategy s) {
datetimeStrategy = s; .. }
    private IDateTime_Strategy datetimeStrategy;
    private DateTime Now { get { return
datetimeStrategy.Now; } }
}
```

Which allows the outgoing dependency on "DateTime.Now" to be stubbed through a C# interface. However, we've now added a dynamic dispatch call where static would suffice, the object is larger even in the production version, and we've added a new failure path for Foo's constructor (allocation can fail).

Am I worrying about nothing here? Thanks for the feedback so far!

unit-testing stub conditional-compilation

Share
Improve this question
Follow

edited May 23, 2017 at 11:57
Community Bot
1 ● 1

asked Sep 18, 2008 at 21:21
Aaron
3,474 ● 25 ● 26

6 Answers

Sorted by: Highest score (default)

3

Try to keep production code separate from test code. Maintain different folder hierarchies.. different solutions/projects.

Unless.. you're in the world of legacy C++ Code. Here anything goes.. if conditional blocks help you get some of the code testable and you see a benefit.. By all means do it. But try to not let it get messier than the initial state. Clearly comment and demarcate conditional blocks. Proceed with caution. It is a valid technique for getting legacy code under a test harness.

Share Improve this answer
Follow

answered Sep 18, 2008 at 21:44
Gishu
137k ● 47 ● 226 ● 311

Do you mean "this is okay for existing (legacy) code bases when incrementally adding unit testing" or "this works in C++" or "this works in legacy C++, but shoudl be discouraged when starting a fresh new C++ project"? – Aaron Sep 18, 2008 at 23:26

First clause in your OR sequence. Its a nice technique to get legacy code under test without disturbing the code-base too much (You're not confident of making these changes because you don't have the safety net of tests). – Gishu Sep 19, 2008 at 4:31

2

I think it lessens the clarity for people reviewing the code. You shouldn't have to remember that there's a conditional tag around specific code to understand the context.

Share Improve this answer Follow

answered Sep 18, 2008 at 21:27

plyawn
412 ● 4 ● 13

1

No this is terrible. It leaks test into your production code (even if its conditioned off)

Bad bad.

Share Improve this answer Follow

answered Sep 18, 2008 at 21:34

Aaron Jensen
6,060 ● 1 ● 32 ● 40

1

Test code should be obvious and not inter-mixed in the same blocks as the tested code.

This is pretty much the same reason you shouldn't write

```
if (globals.isTest)
```

answered Sep 18, 2008 at 21:36

Amy B
110k ● 21 ● 139 ● 190

1

I thought of another reason this was terrible:

Many times you mock/stub something, you want its methods to return different results depending on what you're testing. This either precludes that or makes it awkward as all heck.

answered Sep 19, 2008 at 14:48

Aaron Jensen
6,060 ● 1 ● 32 ● 40

How is it any more awkward than using a C# interface? In both cases, you have one method implementation body for mocking, and one method implementation body for production behavior. – Aaron Sep 22, 2008 at 17:43

Have you used a mocking framework? You don't have additional bodies for testing. Furthermore, I'm talking about situations where you want, say, 7 different behaviors depending on your context. It's just a horrendous idea. Don't do it. – Aaron Jensen Sep 22, 2008 at 19:50

0

It might be useful as a tool to lean on as you refactor to testability in a large code base. I can see how you might use such techniques to enable smaller changes and avoid a "big bang" refactoring. However I would worry about leaning too hard on such a technique and would try to ensure that such tricks didn't live too long in the code base otherwise you risk making the application code very complex and hard to follow.

Share Improve this answer Follow

answered Sep 18, 2008 at 21:36

Bradley Harris
932 ● 1 ● 6 ● 12