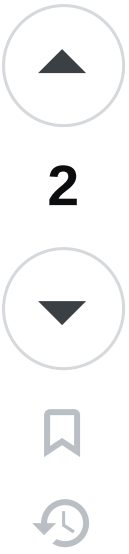
What is the appropriate amount of error-checking?

Asked 16 years, 2 months ago Modified 16 years, 2 months ago

Viewed 346 times



```
public void PublicMethod(FooBar fooBar)
{
    if (fooBar == null)
        throw new ArgumentNullException("fooBar",
"fooBar cannot be null");

    // log the call [added:  Thanks S.Lott]
    _logger.Log("PublicMethod called with fooBar
class " + fooBar.Classification);

    int action = DetermineAction();
    PrivateMethod(fooBar, action);
}

private void PrivateMethod(FooBar fooBar, int
action)
{
    if (fooBar == null)
        throw new ArgumentNullException("fooBar",
"fooBar cannot be null");  // Is this line
superfluous?

    /*
        Do something
    */
}
```

Is it OK to skip this kind of error checking in private methods if the input is already checked on the public

interface? Usually there's some sort of rule-of-thumb one can go by...

Edit:

Maybe ArgumentNullException isn't such a good example because the argument can be made that you should check at both levels but return different error messages.

error-checking

Share

Improve this question

Follow

edited Oct 8, 2008 at 15:07

asked Oct 8, 2008 at 14:47

ilitirit

16.3k ● 18 ● 77 ● 115

Java? C#? Language-Agnostic? – S.Lott Oct 8, 2008 at 14:54

9 Answers

Sorted by: Highest score (default)

I would say no.

While it certainly holds true that you in *this* case knows that it has already been checked for nullability, in two months time the youngest intern will come along and write PublicMethod2 that also calls PrivateMethod, but lo and behold he forgot to check for null.

Share Improve this answer

Follow

answered Oct 8, 2008 at 14:50

Soraz

6,736 ● 4 ● 33 ● 49

I disagree. In the case you describe, the problem is in PublicMethod2, not in PrivateMethod. – James Curran Oct 8, 2008 at 15:05

Since the public method doesn't really use foobar, I'm not sure why it's checking. The current private method cares, but it's the private method's responsibility to care. Indeed, the whole point of a private method is to delegate all the responsibilities to it.

A method checks the input it actually uses; it doesn't check stuff it's just passing through.

If a different subclass has the same public method, but some different private method implementation -- one that can tolerate nulls -- what now? You have a public method that now has wrong constraints for the new subclass.

You want to do as little as possible in the public method so that various private implementations are free to do the right thing. Don't "over-check" or "just-in-case" check. Delegate responsibility.

Share Improve this answer

Follow

edited Oct 8, 2008 at 15:56

answered Oct 8, 2008 at 15:00

S.Lott

391k ● 82 ● 517 ● 788

> A good point. I'll adjust the example, but it seems that you're saying a method doesn't need to check inputs it doesn't use, or are you saying that a method should check all the inputs it uses? – ilitirit Oct 8, 2008 at 15:14

1

I'd error check everything you can, you never know when something might happen that you didn't think about. (and its better safe than sorry)

Share Improve this answer

Follow

answered Oct 8, 2008 at 15:09

Miles

5,736 ● 19 ● 65 ● 87

1

When using design by contract (http://en.wikipedia.org/wiki/Design_by_contract) it's normally client's (public method) responsibility to make



correct invocation, i.e. pass on valid parameters. In this particular scenario it depends whether null belongs to a set of valid input values, therefore there are 3 options:

1) Null is valid value: throwing exceptions or errors would have meant breaking the contract, the server (private method) has to process the null and shouldn't complain.

2) Null is invalid value and passed by code within your control: it is up to the server (private method) to decide how to react. Obviously, throwing an exception is more graceful way of handling the situation, but it has a cost of having to handle that exception somewhere else up the stack. Exceptions are not the best way to deal with violation of contract caused by programming blunders. You really should throw exceptions not when a contract is already violated but when it cannot be fulfilled because of environmental problems what cannot be controlled in software. Blunders are better handled by sticking an assertion into the beginning of the private method to check that the parameter is not null. This will keep the complexity of your code down, there is no cost of having to handle the exception up the stack and it will achieve the goal of highlighting broken contracts during testing.

3) Then there is defensive programming (http://en.wikipedia.org/wiki/Defensive_programming). When dealing with parameters passed by an external code outside your control the immediate layer of your code needs to run paranoid level of checks and return errors according to its communication contract with the

external world. Then, going deeper into the code layers not exposed externally, it still makes more sense to stick to the programming by contract.

Share Improve this answer

Follow

answered Oct 8, 2008 at 16:10

Vlad Gudim

23.5k ● 16 ● 71 ● 92

0

At least put a comment that PrivateMethod must have a non-null FooBar and that PublicMethod checks this.

Share Improve this answer

Follow

answered Oct 8, 2008 at 14:49

Robert Deml

12.5k ● 21 ● 69 ● 92

0

You might want to also mark the "private" method as private or protected.

Share Improve this answer

Follow

answered Oct 8, 2008 at 14:54

florin

14.3k ● 6 ● 49 ● 47

That depends if a null-value indicates an error for a method. Remember that methods could also be called

0

messages to an object; they operate on the state of the object aswell. Parameters can specialize the kind of message sent.

- If publicMethod() does not use a parameter and changes the state of the instance while privateMethod() uses the parameter, do not consider it an error in publicMethod, but do in privateMethod().
- If publicMethod() does not change state, consider it an error.

You could see the latter case as providing an interface to the internal functioning of an object.

Share Improve this answer Follow

answered Oct 8, 2008 at 16:05

mstrobl

2,391 ● 15 ● 16

0

I'd consider the answer to be "yes, do the check again" because:-

- The private member could be reused again in the future from a different path through the code, so program defensively against that situation.
- If you perform unit tests on private methods

My view might change if I had a static analyser that could pick this up and not flag the potential use of a null reference in the private method.

answered Oct 8, 2008 at 16:32

Rhys
4,579 ● 2 ● 25 ● 32

In cases where PrivateMethod will be called frequently with input that has already been verified, and only rarely with user input, Then I would use the PublicMethod/PrivateMethod concept with no error checking on PrivateMethod (and with PublicMethod doing nothing other then checking the parameters and calling PrivateMethod)

I would also call the private method something like PublicMethod_impl (for "implementation") so it's clear that it's an internal use/ no checking method.

I maintain that this design leads to *more* robust application, as it forces you to think about what's checked when. Too often people who always check parameters fall into the trap of "I've checked something, therefore I've checked everything".

As an example of this, a former co-worker (programming in C) would, before using a pointer, always check to see if it was null. Generally, the pointers in his code were initialized as startup and never changed, so the chances of it being null were quite low. Moreover, the pointer has one correct value and 65535 possible wrong values, and he was only checking for one of those wrong values.

answered Oct 8, 2008 at 17:15

Follow

James Curran

103k 37 185 262