# Debug.Assert vs. Specific Thrown Exceptions [duplicate]

Asked 16 years, 3 months ago Modified 13 years, 7 months ago Viewed 10k times

32

**This question already has answers here**:

Debug.Assert vs Exception Throwing (8 answers)

Closed 2 months ago.

I've just started skimming 'Debugging MS .Net 2.0 Applications' by John Robbins, and have become confused by his evangelism for Debug.Assert(...).

He points out that well-implemented Asserts store the state, somewhat, of an error condition, e.g.:

```
Debug.Assert(i > 3, "i > 3", "This means I got a bad parameter");
```

Now, personally, it seems crazy to me that he so loves restating his test without an actual sensible 'business logic' comment, perhaps "i <= 3 must never happen because of the flobittyjam widgitification process".

So, I think I get Asserts as a kind-of low-level "Let's protect my assumptions" kind of thing... assuming that one feels this is a test one only needs to do in debug - i.e. you are protecting yourself against colleague and future programmers, and hoping that they actually test things.

But what I don't get is, he then goes on to say that you should use assertions in addition to normal error handling; now what I envisage is something like this:

```
Debug.Assert(i > 3, "i must be greater than 3 because of the flibbity widgit
status");
if (i <= 3)
{
    throw new ArgumentOutOfRangeException("i", "i must be > 3 because... i=" +
i.ToString());
}
```

What have I gained by the Debug.Assert repetition of the error condition test? I think I'd get it if we were talking about debug-only double-checking of a very important calculation...

```
double interestAmount = loan.GetInterest();
Debug.Assert(debugInterestDoubleCheck(loan) == interestAmount, "Mismatch on
interest calc");
```

...but I don't get it for parameter tests which are surely worth checking (in both DEBUG and Release builds)... or not. What am I missing?

c# exception assert

Share Improve this question Follow

asked Sep 14, 2008 at 9:19

Nij
2,068 ● 2 ● 23 ● 27

## 8 Answers

Sorted by: Highest score (default)

51

Assertions are not for parameter checking. Parameter checking should always be done (and precisely according to what pre-conditions are specified in your documentation and/or specification), and the `ArgumentOutOfRangeException` thrown as necessary.

Assertions are for testing for "impossible" situations, i.e., things that you (in your program logic) *assume* are true. The assertions are there to tell you if these assumptions are broken for any reason.

Hope this helps!

Share Improve this answer Follow

answered Sep 14, 2008 at 9:26

C. K. Young
223k ● 47 ● 390 ● 443

10 Assertions can be used for parameter checking of *internal* method calls (called by code belonging to the same component) methods, as opposed to external method calls (called by another component). For example, I might assert that a private method parameter of type Double isn't a NaN. – HTTP 410 Oct 31, 2008 at 11:48

@Chris: You state that Assertions are not to be used for parameter checking. Is there a reason for this? I tend to throw exceptions for parameter checks especially in constructors when I inject dependent objects. However I'm being told to use Assertions. I don't have a logical explanation to using exceptions rather than assertions. Are you able to clarify? Cheers – Gavin Chin Sep 23, 2009 at 6:41

7 Don't worry I found out why. According to Jon Skeet, stackoverflow.com/questions/1276308/exception-vs-assertion: "Use assertions for internal

logic checks within your code, and normal exceptions for error conditions outside your immediate code's control." – Gavin Chin Sep 23, 2009 at 7:06

1 I have noticed in my years of development, that assertions are often only possible to make when you have too much data. (e.g. redundancy of state storage). Which by definition is not too good. Shorter and better code may not allow to have a way to check for coherency. For example "if my bool bIsReceiving is true then i32 iSourceIP must be non null". This is an example of a useless boolean, you stored two times the same information. just make a function IsReceiving() returning iSourceIP != null. my 2 cts. – v.oddou Feb 22, 2013 at 2:37

---

17

There is a communication aspect to asserts vs exception throwing.

Let's say we have a User class with a Name property and a ToString method.

If ToString is implemented like this:

```
public string ToString()
{
     Debug.Assert(Name != null);
     return Name;
}
```

It says that Name should never null and there is a bug in the User class if it is.

If ToString is implement like this:

```
public string ToString()
{
     if ( Name == null )
     {
          throw new InvalidOperationException("Name is null");
     }

     return Name;
}
```

It says that the caller is using ToString incorrectly if Name is null and should check that before calling.

The implementation with both

```
public string ToString()
{
     Debug.Assert(Name != null);
     if ( Name == null )
     {
          throw new InvalidOperationException("Name is null");
     }
```

```
        return Name;
}
```

says that if Name is null there bug in the User class, but we want to handle it anyway. (The user doesn't need to check Name before calling.) I think this is the kind of safety Robbins was recommending.

Share Improve this answer Follow

answered Sep 14, 2008 at 10:57

hwiechers

15k ● 8 ● 57 ● 62

7

I've thought about this long and hard when it comes to providing guidance on debug vs. assert with respect to testing concerns.

You should be able to test your class with erroneous input, bad state, invalid order of operations and any other conceivable error condition and an assert should **never** trip. Each assert is checking something should **always** be true regardless of the inputs or computations performed.

Good rules of thumb I've arrived at:

1. Asserts are not a replacement for robust code that functions correctly independent of configuration. They are complementary.
2. Asserts should never be tripped during a unit test run, even when feeding in invalid values or testing error conditions. The code should handle these conditions without an assert occurring.
3. If an assert trips (either in a unit test or during testing), the class is bugged.

For all other errors -- typically down to environment (network connection lost) or misuse (caller passed a null value) -- it's much nicer and more understandable to use hard checks & exceptions. If an exception occurs, the caller knows it's likely their fault. If an assert occurs, the caller knows it's likely a bug in the code where the assert is located.

Regarding duplication: I agree. I don't see why you would replicate the validation with a Debug.Assert AND an exception check. Not only does it add some noise to the code and muddy the waters regarding who is at fault, but it a form of repetition.

Share Improve this answer Follow

answered Jan 9, 2010 at 16:06

Mark Simpson

23.4k ● 2 ● 46 ● 44

4

I use explicit checks that throw exceptions on *public* and *protected* methods and assertions on private methods.

Usually, the explicit checks guard the private methods from seeing incorrect values anyway. So really, the assert is checking for a condition that should be impossible. If an assert does fire, it tells me the there is a defect in the validation logic contained within one of the public routines on the class.

Share Improve this answer Follow

answered Sep 14, 2008 at 9:55



Simon Johnson

**7,902** ● 5 ● 38 ● 50

---

3

An exception can be caught and swallowed making the error invisible to testing. That can't happen with Debug.Assert.

No one should ever have a catch handler that catches all exceptions, but people do it anyway, and sometimes it is unavoidable. If your code is invoked from COM, the interop layer catches all exceptions and turns them into COM error codes, meaning you won't see your unhandled exceptions. Asserts don't suffer from this.

Also when the exception would be unhandled, a still better practice is to take a mini-dump. One area where VB is more powerful than C# is that you can use an exception filter to snap a mini-dump when the exception is in flight, and leave the rest of the exception handling unchanged. Gregg Miskelly's blog post on exception filter inject provides a useful way to do this from c#.

One other note on assets ... they inteact poorly with Unit testing the error conditions in your code. It is worthwhile to have a wrapper to turn off the assert for your unit tests.

Share

Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

**1** ● 1

answered Sep 14, 2008 at 21:59

Steve Steiner

**5,339** ● 4 ● 33 ● 45

---

2

IMO it's a loss of development time only. Properly implemented exception gives you a clear picture of what happened. I saw *too much* applications showing obscure "Assertion failed: i < 10" errors. I see assertion as a temporary solution. In my opinion no assertions should be in a final version of a program. In my practice I used assertions for quick and dirty checks. Final version of the code should take erroneous situation into account and behave accordingly. If something bad happens you have 2 choices: handle it or leave it. Function should throw an exception with meaningful description if wrong parameters passed in. I see no points in duplication of validation logic.

Share

Improve this answer

Follow

edited Sep 14, 2008 at 9:32

answered Sep 14, 2008 at 9:25

aku

124k ● 33 ● 176 ● 203

---

1

Example of a good use of Assert:

```
Debug.Assert(flibbles.count() < 1000000, "too many flibbles"); // indicate
something is awry
log.warning("flibble count reached " + flibbles.count()); // log in production
as early warning
```

I personally think that Assert should **only** be used when you know something is outside *desirable* limits, but you can be sure it's reasonably safe to continue. In all other circumstances (feel free point out circumstances I haven't thought of) use exceptions to fail hard and fast.

The key tradeoff for me is whether you want to bring down a live/production system with an Exception to avoid corruption and make troubleshooting easier, or whether you have encountered a situation that should never be allowed to continue unnoticed in test/debug versions but could be allowed to continue in production (logging a warning of course).

cf. http://c2.com/cgi/wiki?FailFast copied and modified from java question: Exception Vs Assertion

Share

Improve this answer

Follow

edited May 23, 2017 at 12:13

Community Bot

1 ● 1

answered May 11, 2011 at 12:31

Tim Abell

11.9k ● 9 ● 87 ● 112

---

0

Here is by 2 cents.

I think that the best way is to use both assertions and exceptions. The main differences between the two methods, imho, if that Assert statements can be removed easily from the application text (defines, conditional attributes...), while Exception thrown are dependent (tipically) by a conditional code which is harder to remove (multine section with preprocessor conditionals).

Every application exception shall be handled correctly, while assertions shall be satisfied only during the algorithm developement and testing.

If you pass an null object reference as routine parameter, and you use this value, you get a null pointer exception. Indeed: why you should write an assertion? It's a waste of time in this case. But what about private class members used in class routines? When

these value are set somewhere, is better to check with an assertion if a null value is set. That's only because when you use the member, you get a null pointer exception but you don't know how the value was set. This cause a restart of the program breaking on all entry point use to set the private member.

Exception are more usefull, but they can be (imho) very heavy to manage and there is the possibility to use too much exceptions. And they requires additional check, maybe undesired to optimize the code. Personally I use exceptions only whenever the code requires a deep catch control (catch statements are very low in the call stack) or whenever the function parameters are not hardcoded in the code.



answered Jan 9, 2010 at 15:22

Luca

**12k** ● 12 ● 73 ● 125