

Software Engineering Stack Exchange is a question and answer site for professionals, academics, and students working within the systems development life cycle. It only takes a minute to sign up.

[Sign up to join this community](#)

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top



SPONSORED BY  **stackoverflow**
FOR TEAMS

Are there guidelines on how many parameters a function should accept?

Asked 7 years, 5 months ago Active 8 months ago Viewed 97k times



114



44

I've noticed a few functions I work with have 6 or more parameters, whereas in most libraries I use it is rare to find a function that takes more than 3.

Often a lot of these extra parameters are binary options to alter the function behaviour. I think that some of these umpteen-parametered functions should probably be refactored. Is there a guideline for what number is too many?

[functions](#)

[parameters](#)

edited Mar 13 '17 at 13:31



[jchanger](#)

103 4


asked Apr 18 '12 at 17:31



[Darth Egregious](#)

679 2 7 8

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

-
- 2 Definitely don't follow the example of [MPI_Sendrecv\(\)](#), which takes 12 parameters! – [chrisaycock](#) Apr 18 '12 at 20:57 
-
- 6 The project I'm currently working on uses a certain framework, in which methods with 10+ parameters is commonplace. I'm calling one particular method with 27 parameters in a couple of places. Every time I see it I die a little inside. – [perp](#) Apr 19 '12 at 5:30
-
- 3 Never add boolean switches to alter function behavior. Split the function instead. Break out the common behavior into new functions. – [kevin cline](#) Feb 9 '16 at 0:03
-
- 2 @Ominus What? Only 10 parameters? That's nothing, [much more is needed](#). :D – [maaartinus](#) Nov 4 '17 at 1:38
-

11 Answers



106



I've never seen a guideline, but in my experience a function that takes more than three or four parameters indicates one of two problems:

1. The function is doing too much. It should be split into several smaller functions, each which have a smaller parameter set.
2. There is another object hiding in there. You may need to create another object or data structure that includes these parameters. See this article on the [Parameter Object](#) pattern for more information.

It's difficult to tell what you're looking at without more information. Chances are the refactoring you need to do is split the function into smaller functions which are called from the parent depending on those flags that are currently being passed to the function.

There are some good gains to be had by doing this:

- It makes your code easier to read. I personally find it much easier to read a "rules list" made up of an `if` structure that calls a lot of methods with descriptive names than a structure that does it all in one method.
- It's more unit testable. You've split your problem into several smaller tasks that are individually very simple. The unit test collection would then be made up of a behavioral test suite that checks the paths through the master method and a collection of smaller tests for each individual procedure.

edited Sep 27 '17 at 17:47



[Robert Harvey](#)

173k 47 408 615

answered Apr 18 '12 at 18:09



[Michael K](#)

13k 8 52 92

-
- 5 Parameter abstraction has been turned into a design pattern? What happens if you have 3 parameter classes. Do you add 9 more method overloads to

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

Unless you use the Pattern Object pattern to package variables in an object instance and pass that in as a parameter. That works for packaging but it may be tempting to create classes of dissimilar variables just for the convenience of simplifying the method definition. – [Evan Plaice](#) Apr 18 '12 at 19:14

@EvanPlaice I'm not saying that you have to use that pattern whenever you have more than one parameter - you're absolutely correct that that becomes even nastier than the first list. There may be cases where you really do need a large number of parameters and it just doesn't work to wrap them up in an object. I've yet to meet a case in enterprise development that did not fall into one of the two buckets I mentioned in my answer - that's not saying one doesn't exist though. – [Michael K](#) Apr 18 '12 at 19:14

@MichaelK If you have never used them, try googling 'object initializers'. It's a pretty novel approach that significantly reduces definition boilerplate. In theory, you could eliminate a classes constructors, parameters, and overloads all in one shot. In practice though, it's usually good to maintain one common constructor and rely on the 'object initializer' syntax for the rest of the obscure/niche properties. IMHO, it's the closest you get to the expressiveness of dynamically typed languages in a statically typed language. – [Evan Plaice](#) Apr 18 '12 at 20:20

@Evan Plaice: Since when are dynamically typed languages expressive? – [ThomasX](#) Apr 25 '12 at 8:24



According to "Clean Code: A Handbook of Agile Software Craftsmanship", zero is the ideal, one or two are acceptable, three in special cases and four or more, never!

41

The words of the author:



The ideal number of arguments for a function is zero (niladic). Next comes one (monadic), followed closely by two (dyadic). Three arguments (triadic) should be avoided where possible. More than three (polyadic) requires very special justification—and then shouldn't be used anyway.

In this book there is a chapter talking only about functions where parameters are large discussed, so I think this book can be a good guideline of how much parameters you need.

In my personal opinion, one parameter is better than no one because I think is more clear what is going on.

As example, in my opinion the second choice is better because is more clear what the method is processing:

```
LangDetector detector = new LangDetector(someText);  
//lots of lines  
String language = detector.detectLanguage();
```

VS.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

About a lot of parameters, this can be a sign that some variables can be grouped into a single object or, in this case, a lot of booleans can represent that the function/method is doing more than one thing, and in this case, is better refactoring each one of these behaviors in a different function.

edited Apr 19 '12 at 10:27

answered Apr 18 '12 at 20:01



[Renato Dinhani](#)

1,302 1 11 18

-
- 8 "three in special cases and four or more, never!" BS. How about `Matrix.Create(x1,x2,x3,x4,x5,x6,x7,x8,x9)`; ? – [Lukasz Madon](#) Apr 19 '12 at 2:26
-
- 71 Zero is ideal? How the heck do functions get information? Global/instance/static/whatever variables? YUCK. – [Peter C](#) Apr 19 '12 at 3:13
-
- 9 That's a bad example. The answer is obviously: `String language = detectLanguage(someText);` . In either of your cases you passed the exact same number of arguments, it just happens that you've split the function execution in two because of a poor language. – [Matthieu M.](#) Apr 19 '12 at 6:20
-
- 8 @lukas, in languages supporting such fancy constructs as arrays or (gasp!) lists, how about `Matrix.Create(input)`; where `input` is, say, a `.NET IEnumerable<SomeAppropriateType>` ? That way you also don't need a separate overload when you want to create a matrix holding 10 elements instead of 9. – [a CVn](#) Apr 19 '12 at 7:49
-
- 9 Zero arguments as "ideal" is a crock and one reason I think Clean Code is way overrated. – [user949300](#) Sep 28 '17 at 2:35
-

▲ If the domain classes in the application are designed correctly, the number of parameters that we pass into a function will be automatically reduced - because the classes know how to do their job and they have enough data to do their work.

24



For example, say you have a manager class which asks a 3rd grade class to complete the assignments.

If you model correctly,

```
3rdGradeClass.finishHomework(int lessonId) {
    result = students.assignHomework(lessonId, dueDate);
    teacher.verifyHomeWork(result);
}
```

This is simple.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
    // This is not good.  
}
```

The correct model always reduces the function parameters between the method calls as the correct functions are delegated to their own classes (Single responsibility) and they have enough data to do their work.

Whenever I see the number of parameters increasing, I check my model to see if I designed my application model correctly.

There are some exceptions though: When I need to create a transfer object or config objects, I will use a builder pattern to produce small built objects first before constructing a big config object.

edited Sep 13 '16 at 15:04

answered Apr 18 '12 at 18:59



java_mouse

2,289 11 23



One aspect that the other answers do not take up is performance.

16



If you are programming in a sufficiently low level language (C, C++, assembly) a large number of parameters can be quite detrimental to performance on some architectures, especially if the function is called a large amount of times.

When a function call is made in ARM for instance, the first four arguments are placed in registers `r0` to `r3` and remaining arguments have to be pushed onto the stack. Keeping the number of arguments below five can make quite a difference for critical functions.

For functions that are called extremely often, even the fact that the program has to set up the arguments before each call can affect performance (`r0` to `r3` may be overwritten by the called function and will have to be replaced before the next call) so in that regard zero arguments are best.

Update:

KjMag brings up the interesting topic of inlining. Inlining will in some ways mitigate this since it will allow the compiler to perform the same optimizations that you would be able to do if writing in pure assembly. In other words, the compiler can see which parameters and variables are used by the called function and can optimize register usage so that stack read/write is minimized.

There are some problems with inlining though.

- 1 Inlining causes the compiled binary to grow since the same code is duplicated in binary form if it is called from multiple places. This

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

from an inlined function. Binary growth would explode if `inline` was treated as mandatory in all cases.

3. There are plenty of compilers that will either completely ignore `inline` or actually give you errors when they encounter it.

edited Nov 4 '17 at 1:07

answered Apr 19 '12 at 14:38



Leo

260 1 8

Whether passing a large number of parameters is good or bad from a performance perspective depends upon the alternatives. If a method needs a dozen pieces of information, and one will be calling it hundreds of times with the same values for eleven of them, having the method take an array could be faster than having it take a dozen parameters. On the other hand, if every call will need a unique set of twelve values, creating and populating the array for each call could easily be slower than simply passing the values directly. – [supercat](#) Sep 2 '14 at 21:44

Doesn't inlining solve this problem? – [KjMag](#) Nov 3 '17 at 10:51

@KjMag: Yes, to a certain degree. But there are a lot of gotchas depending on the compiler. Functions will usually only be inlined up to a certain level (if you call an inlined function that calls an inlined function that calls an inlined function....). If the function is large and is called from a lot of places, inlining everywhere makes the binary larger which may mean more misses in the L-cache. So inlining can help, but it's no silver bullet. (Not to mention there are quite a few old embedded compilers that don't support `inline`.) – [Leo](#) Nov 4 '17 at 0:50



When the parameter list grows to more than five, consider defining a "context" structure or object.

7

This is basically just a structure that holds all the optional parameters with some sensible defaults set.



In the C procedural world a plain structure would do. In Java, C++ a simple object will suffice. Don't mess with getters or setters because the sole purpose of the object is to hold "public"ly settable values.

answered Apr 19 '12 at 1:39



James Anderson

16.8k 1 34 64

I agree, a context object can turn out to be very handy when function parameter structures start becoming rather complex. I had recently blogged about using [context objects with a visitor-like pattern](#) – [Lukas Eder](#) Apr 19 '12 at 11:56

No, there is no standard guideline

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

**You could use a list-if-args parameter (args*) or a dictionary-of-args parameter (kwargs **)**

For instance, in python:

```
// Example definition
def example_function(normalParam, args*, kwargs**):
    for i in args:
        print 'args' + i + ': ' + args[i]
    for key in kwargs:
        print 'keyword: %s: %s' % (key, kwargs[key])
    somevar = kwargs.get('somevar', 'found')
    missingvar = kwargs.get('somevar', 'missing')
    print somevar
    print missingvar

// Example usage

    example_function('normal parameter', 'args1', args2,
                    somevar='value', missingvar='novalue')
```

Outputs:

```
args1
args2
somevar:value
someothervar:novalue
value
missing
```

Or you could use object literal definition syntax

For example, here's a JavaScript jQuery call to launch an AJAX GET request:

```
$.ajax({
    type: 'GET',
    url: 'http://someurl.com/feed',
    data: data,
    success: success(),
    error: error(),
    complete: complete(),
    dataType: 'json'
})
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

If you take a look at jQuery's ajax class there are a *lot* (approximately 30) more properties that can be set; mostly because ajax communications are very complex. Fortunately, the object literal syntax makes life easy.

C# intellisense provides active documentation of parameters so it's not uncommon to see very complex arrangements of overloaded methods.

Dynamically typed languages like python/javascript have no such capability, so it's a lot more common to see keyword arguments and object literal definitions.

I prefer object literal definitions ([even in C#](#)) for managing complex methods because you can explicitly see which properties are being set when an object is instantiated. You'll have to do a little more work to handle default arguments but in the long run your code will be a lot more readable. With object literal definitions you can break your dependence on documentation to understand what your code is doing at first glance.

IMHO, overloaded methods are highly overrated.

Note: If I remember right readonly access control should work for object literal constructors in C#. They essentially work the same as setting properties in the constructor.

If you have never written any non-trivial code in a dynamically typed (python) and/or functional/prototype javaScript based language, I highly suggest trying it out. It can be an enlightening experience.

It's can be scary first to break your reliance on parameters for the end-all, be-all approach to function/method initialization but you will learn to do so much more with your code without having to add unnecessary complexity.

Update:

I probably should have provided examples to demonstrate use in a statically typed language but I'm not currently thinking in a statically typed context. Basically, I've been doing too much work in a dynamically typed context to suddenly switch back.

What I do know is object literal definition syntax is completely possible in statically typed languages (at least in C# and Java) because I have used them before. In statically typed languages they're called 'Object Initializers'. Here are some links to show their use in [Java](#) and [C#](#).

[edited May 23 '17 at 12:40](#)

[answered Apr 18 '12 at 18:25](#)

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

- 3 I'm not sure I like this method, mainly because you lose the self documenting value of individual parameters. For lists of like items this makes perfect sense (for instance a method that takes a list of strings and concatenates them) but for an arbitrary parameter set this is worse than the long method call. – [Michael K](#) Apr 18 '12 at 18:37

@MichaelK Take another look at object initializers. They allow you to explicitly define properties as opposed to how they're implicitly defined in traditional method/function parameters. Read this, msdn.microsoft.com/en-us/library/bb397680.aspx, to see what I mean. – [Evan Plaice](#) Apr 18 '12 at 18:55

- 3 Creating a new type just to handle the parameter list sounds exactly like the definition of unnecessary complexity... Sure, dynamic languages let you avoid that but then you get a single ball of goo parameter. Regardless, this doesn't answer the question asked. – [Telastyn](#) Apr 18 '12 at 19:11

@Telastyn What are you talking about? No new type was created, you declare properties directly using object literal syntax. It's like defining an anonymous object but the method interprets it as a key=value parameter grouping. What you're looking at is a method instantiation (not a parameter encapsulating object). If your beef is with parameter packaging, take a look at the Parameter Object pattern mentioned in one of the other questions because that's exactly what it is. – [Evan Plaice](#) Apr 18 '12 at 19:18

@EvanPlaice - Except that static programming languages by-and-large require a (often new) declared type in order to allow the Parameter Object pattern. – [Telastyn](#) Apr 18 '12 at 19:30



3

Personally, more than 2 is where my code smell alarm triggers. When you consider functions to be **operations** (that is, a translation from input to output), it is uncommon that more than 2 parameters are used in an operation. *Procedures* (that is a series of steps to achieve a goal) will take more inputs and are sometimes the best approach, but in most languages these days should not be the norm.



But again, that's the guideline rather than a rule. I often have functions that take more than two parameters due to unusual circumstances or ease of use.

answered Apr 18 '12 at 18:09



[Telastyn](#)

96k 27 216 331



2

Very much like Evan Plaice is saying, I'm a huge fan of simply passing associative arrays (or your language's comparable data structure) into functions whenever possible.



Thus, instead of (for example) this:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```
?>
```

Go for:

```
<?php

// create a hash of post data
$post_data = array(
    'title'    => 'the title',
    'summary'  => 'the summary',
    'author'   => 'the author',
    'pubdate'  => 'the publication date',
    'keywords' => 'the keywords',
    'category' => 'the category',
    'etc'      => 'etc',
);

// and pass it to the appropriate function
createBlogPost($post_data);

?>
```

Wordpress does a lot of stuff this way, and I think it works well. (Though my example code above is imaginary, and is not itself an example from Wordpress.)

This technique allows you to pass a lot of data into your functions easily, yet frees you from having to remember the order in which each must be passed.

You'll also appreciate this technique when comes time to refactor - instead of having to potentially change the order of a function's arguments (such as when you realize you need to pass Yet Another Argument), you don't need to change your functions's parameter list at all.

Not only does this spare you from having to re-write your function definition - it spares you from having to change the order of arguments each time the function is invoked. That's a huge win.

answered Apr 24 '12 at 19:26



[Chris Allen Lane](#)

119 4

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

▲ 0 A previous [answer](#) mentioned a reliable author who stated that the less parameters your functions have, the better you are doing. The answer did not explain why but the books explains it, and here are two of the most convincing reasons as why you need to adopt this philosophy and with which I personally agree with:

- Parameters belong to a level of abstraction which is different from that of the function. This means the reader of your code will have to think about the nature and the purpose of the parameters of your functions: this thinking is "lower level" than that of the name and the purpose of their corresponding functions.
- The second reason to have as less parameters as possible to a function is testing: for example, if you have a function with 10 parameters, think about how many combinations of parameters you have to cover all the test cases for, for instance, a unit test. Less parameters = less tests.

answered Sep 27 '17 at 17:23

[Billal Begueradj](#)



704 4 12 29

▲ 0 To provide some more context around the advice for the ideal number of function arguments being zero in Robert Martin's "Clean Code: A Handbook of Agile Software Craftsmanship", the author says the following as one of his points:

Arguments are hard. They take a lot of conceptual power. That's why I got rid of almost all of them from the example. Consider, for instance, the `StringBuffer` in the example. We could have passed it around as an argument rather than making it an instance variable, but then our readers would have had to interpret it each time they saw it. When you are reading the story told by the module, `includeSetupPage()` is easier to understand than `includeSetupPageInto(newPageContent)`. The argument is at a different level of abstraction that the function name and forces you to know a detail (in other words, `StringBuffer`) that isn't particularly important at that point.

For his `includeSetupPage()` example above, here's a small snippet of his refactored "clean code" at the end of the chapter:

```
// *** NOTE: Comments are mine, not the author's ***
//
// Java example
public class SetupPageInclude {
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

// this is the zero-argument function in the example,
// which calls a method that eventually uses the StringBuffer instance variable
private void includeSetupPage() throws Exception {
    include("SetUp", "-setup");
}

private void include(String pageName, String arg) throws Exception {
    WikiPage inheritedPage = findInheritedPage(pageName);
    if (inheritedPage != null) {
        String pagePathName = getPathNameForPage(inheritedPage);
        buildIncludeDirective(pagePathName, arg);
    }
}

private void buildIncludeDirective(String pagePathName, String arg) {
    newPageContent
        .append("\n!include ")
        .append(arg)
        .append(" .")
        .append(pagePathName)
        .append("\n");
}
}

```

The author's "school of thought" argues for small classes, low (ideally 0) number of function arguments, and very small functions. While I also don't fully agree with him, I found it thought-provoking and I feel that the idea of zero function arguments as an ideal can be worth considering. Also, note that even his small code snippet above has non-zero argument functions as well, so I think it depends on the context.

(And as others have pointed out, he also argues that more arguments make it harder from a testing point of view. But here I mainly wanted to highlight the above example and his rationale for zero function arguments.)

edited Feb 7 at 19:04

answered Feb 7 at 18:58



sonny

139 1 6



Ideally zero. One or two are ok, three in certain cases.
Four or more is usually a bad practice.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

If there is one parameter, knowing it's values, testing them and finding error with them is 'relatively easy as there is only one factor. As you increase the factors, the total complexity increases rapidly. For an abstract example:

Consider a 'what to wear in this weather' program. Consider what it could do with one input - temperature. As you can imagine, the results of what to wear are pretty simple based on that one factor. Now consider what the program might/could/should do if it is actually passed temperature, humidity, dewpoint, precipitation, etc. Now imagine how hard it would be to debug if it gave the 'wrong' answer to something.

answered Apr 18 '12 at 20:42



[Michael Durrant](#)

11.4k 3 29 56

12 If a function has zero parameters, it's either returning a constant value (useful in some circumstances, but rather limiting) or it is using some concealed state that would be better off made explicit. (In OO method calls, the context object is sufficiently explicit that it doesn't cause problems.) – [Donal Fellows](#) Apr 18 '12 at 20:55

4 -1 for not quoting the source – [Joshua Drake](#) Apr 19 '12 at 13:31

Are you seriously saying that ideally all functions would take no parameters? Or is this hyperbole? – [GreenAsJade](#) Sep 2 '14 at 11:35 ✎

1 See Uncle Bob's argument at: informit.com/articles/article.aspx?p=1375308 and note that at the bottom he says "Functions should have a small number of arguments. No argument is best, followed by one, two, and three. More than three is very questionable and should be avoided with prejudice." – [Michael Durrant](#) Sep 2 '14 at 11:58

I have given the source. Funny no comments since then. I have also tried to answer the 'guideline' part as many now consider Uncle Bob and Clean Code to be guidelines. Interesting that the hugely upvoted top answer (currently) says not aware of any guideline. Uncle Bob did not *intend* to be authoritative but it somewhat is and this answer at least tries to be an answer to the specifics of the question. – [Michael Durrant](#) Nov 29 '18 at 16:40 ✎

protected by [gnat](#) Mar 17 '14 at 11:53

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 [reputation](#) on this site (the [association bonus](#) does not count).

Would you like to answer one of these [unanswered questions](#) instead?