

What programming practice that you once liked have you since changed your mind about? [closed]

Asked 15 years, 5 months ago Modified 13 years, 3 months ago

Viewed 9k times

99

votes



As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Closed 13 years ago.



Locked. This question and its answers are [locked](#) because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

As we program, we all develop practices and patterns that we use and rely on. However, over time, as our understanding, maturity, and even technology usage changes, we come to realize that some practices that we once thought were great are not (or no longer apply).

An example of a practice I once used quite often, but have in recent years changed, is the use of the [Singleton object pattern](#).

Through my own experience and long debates with colleagues, I've come to realize that [singletons are not always desirable](#) - they can make testing more difficult (by inhibiting techniques like mocking) and can create undesirable coupling between parts of a system. Instead, I now use object factories (typically with a IoC container) that hide the nature and existence of singletons from parts of the system that don't care - or need to know. Instead, they rely on a factory (or service locator) to acquire access to such objects.

My questions to the community, in the spirit of self-improvement, are:

- What programming patterns or practices have you reconsidered recently, and now try to avoid?
- What did you decide to replace them with?

language-agnostic

Share

edited May 23, 2017 at 12:01

community wiki
4 revs, 2 users 63%
LBushkin

48 Answers

Sorted by:

Highest score (default)



1

2

Next

158

votes



```
//Coming out of university, we were taught to
ensure we always had an abundance
//of commenting around our code. But applying that
to the real world, made it
//clear that over-commenting not only has the
potential to confuse/complicate
//things but can make the code hard to follow. Now
I spend more time on
//improving the simplicity and readability of the
code and inserting fewer yet
//relevant comments, instead of spending that time
writing overly-descriptive
//commentaries all throughout the code.
```

Share

edited Jul 7, 2009 at 18:12

community wiki

3 revs, 2 users 96%

Luke Baulch

+1. I was about to post this same answer. I found some of my old programming assignments on an archive disc a few weeks ago. It all looked the same. There was almost a 1:1

ratio of lines of comments to lines of code. – [Michael Moussa](#)
Jul 7, 2009 at 2:21

32 Sounds like you commented *incorrectly*, not too much. Code does not speak for itself. No. It really doesn't. Read the latest NT Insider for a good rant about this. If you think comments will be redundant then you are either wrong or you are doing it wrong. Universities don't teach correct commenting it seems (or bug tracking, or version control... *sigh*). There are way too few comments out there. (and fewer good ones)

– [Thomas](#) Jul 7, 2009 at 11:29

5 Code Complete has good tips on commenting, and the data to back it up. – [Thomas](#) Jul 7, 2009 at 11:38

20 Comments should be used to describe *why* the code does what it does (if it's not obvious), not *what* the code does. A possible exception is a crazy bit twiddling / language hack, like Carmack's magic number 0x5f3759df. – [Chris Simmons](#)
Jul 7, 2009 at 18:17

6 @Thomas: I personally think the problem is that teaching good commenting is not something a university can show students. Almost all programs at schools are one-off things; students don't get to experience looking back at code they wrote a year ago and not understand it at all. Also, lower-level classes teach really simple coding concepts - commenting at this level is almost necessarily tedious, because of what is happening. In other words, it's like trying to teach someone to swim in a wading pool; it's just not the right context for them to understand the motions. – [Dan Lew](#) Jul 7, 2009 at 21:53

117 Single return points.

votes



I once preferred a single return point for each method, because with that I could ensure that any cleanup needed by the routine was not overlooked.



Since then, I've moved to much smaller routines - so the likelihood of overlooking cleanup is reduced and in fact the *need* for cleanup is reduced - and find that early returns reduce the apparent complexity (the nesting level) of the code. Artifacts of the single return point - keeping "result" variables around, keeping flag variables, conditional clauses for not-already-done situations - make the code appear much more complex than it actually is, make it harder to read and maintain. Early exits, and smaller methods, are the way to go.

Share

answered [Jul 6, 2009 at 22:14](#)

community wiki
[Carl Manaster](#)

3 I agree, when combined with data types that automatically clean themselves up, such as `auto_ptr`, `scoped_ptr`, `CComPtr`, etc. – [i_am_jorf](#) Jul 7, 2009 at 2:54

3 Code clean up is what `try { } finally { }` is for – [banjollity](#) Jul 7, 2009 at 9:23

@banjollity: except for languages that don't support `finally { }`. And note that even in languages that support it, `finally { }` is not ALWAYS guaranteed to execute. – [Chris K](#) Jul 20, 2009 at 14:03

1 @banjollity, Chris: In C++, cleanup is what the destructor is for, and except in extreme circumstances (exit(), a destructor throwing an exception during stack unwind, squirrels cutting your power) it is guaranteed to run. – [David Thornley](#) Feb 1, 2010 at 19:10

4 Agreed. *Replace Nested Conditional with Guard Clauses* ftw! – [Jonik](#) Feb 1, 2010 at 20:53 ✎

110

votes



- Trying to code things perfectly on the first try.
- Trying to create perfect OO model before coding.
- Designing everything for flexibility and future improvements.

In one word **overengineering**.

Share

[edited Jul 7, 2009 at 6:10](#)

community wiki

[2 revs](#)

[wuub](#)

6 Wait, I always get it right on the first try. :) – [i_am_jorf](#) Jul 7, 2009 at 2:52

18 The real money's in getting it subtly wrong the first time and letting it out into the wild. Then, when people are used to the gimped version, swoop in with arrogant showmanship and fix the bug/inefficiency to reap extra glory! ;) – [Eric](#) Jul 7, 2009 at 7:15

7 @jeffamaphone - No, only Jon Skeet gets it right the first time.
– [Jordan Parmer](#) Jul 7, 2009 at 13:03

I like the word "overengineering" – [Neilvert Noval](#) Aug 10, 2011 at 6:34

@jeffamaphone - I always get it right on the first try too. But further attempts give what I need :) – [umbr](#) Sep 7, 2011 at 9:56

78
votes



Hungarian notation (both Forms and Systems). I used to prefix everything. `strSomeString` or `txtFoo`. Now I use `someString` and `textBoxFoo`. It's far more readable and easier for someone new to come along and pick up. As an added bonus, it's trivial to keep it consistent -- camelCase the control and append a useful/descriptive name. Forms Hungarian has the drawback of not always being consistent and Systems Hungarian doesn't really gain you much. Chunking all your variables together isn't really that useful - especially with modern IDE's.

Share

answered [Jul 6, 2009 at 21:48](#)

community wiki
[Kenny Mann](#)

What about in dynamically-typed languages, such as Python or JavaScript? I still find it helpful to use Hungarian notation in these languages so that, when looking at variables, I know what type of variable to expect (if there is a type to expect - of course, it would be foolhardy to treat a dynamically typed

language exactly like a statically typed language.) – [Dan Lew](#)

Jul 6, 2009 at 21:57

- 4 I do similar except: `fooTextBox` and `string's` are just hopefully apparent: `numberOfEntries => int`, `isGreat => bool`, etc. – [rball](#)
Jul 6, 2009 at 21:59
-

+1 for getting rid of Hungarian notation. I agree with [rball](#); `fooTextBox`, `fooService`, `fooString` when its really necessary.
– [blu](#) Jul 6, 2009 at 22:02

- 3 @ [wuub](#): I would argue that with proper naming, you shouldn't need to prefix anything. – [Kenny Mann](#) Jul 7, 2009 at 1:11
-

- 4 By the way what you mentioned is not actual hungarian.
– [Antony Carthy](#) Jul 7, 2009 at 9:16
-

67 The "perfect" architecture

votes



I came up with **THE** architecture a couple of years ago.

Pushed myself technically as far as I could so there were 100% loosely coupled layers, extensive use of delegates, and lightweight objects. It was technical heaven.

And it was crap. The technical purity of the architecture just slowed my dev team down aiming for perfection over results and I almost achieved complete failure.

We now have much simpler less technically perfect architecture and our delivery rate has skyrocketed.

Share

[edited Feb 1, 2010 at 19:01](#)

community wiki
2 revs, 2 users 89%
Bruce McLeod

56
votes



The use of caffeine. It once kept me awake and in a glorious programming mood, where the code flew from my fingers with feverous fluidity. Now it does nothing, and if I don't have it I get a headache.

Share

answered Jul 6, 2009 at 21:45

community wiki
Alex

55 You need to drink even more coffee. If that doesn't work, take up smoking. – MusiGenesis Jul 7, 2009 at 0:48

7 Brad: You don't need those when you have Python: xkcd.com/353 – Peter Jul 7, 2009 at 2:36

Nice Christmas Story reference! :-)) – Steve Echols Jul 7, 2009 at 3:20

2 I broke the habit and then picked it up again, several times (This is now my third cycle). There's nothing quite like coding in the cold mornings with a warm mug of coffee! – Matthew Iselin Jul 7, 2009 at 9:33

15 "Looks like I picked the wrong week to quit amphetamines." – ShreevatsaR Jul 11, 2009 at 19:03

49

votes



Commenting out code. I used to think that code was precious and that you can't just delete those beautiful gems that you crafted. I now delete any commented-out code I come across unless there's a TODO or NOTE attached because it's too perilous to leave it in. To wit, I've come across old classes with huge commented-out portions and it really confused me why they were there: were they recently commented out? is this a dev environment change? why does it do this unrelated block?

Seriously consider not commenting out code and just deleting it instead. If you need it, it's still in source control. YAGNI though.

Share

answered [Jul 7, 2009 at 17:16](#)

community wiki
[bbrown](#)

-
- 6 I comment out the old code during refactoring, but only until I verify that the replacement code works. Once the new version is fully functional, I delete the old commented lines.
– [muusbolla](#) Jul 10, 2009 at 17:45

Indeed - I also comment out code, but only for a few days. If I come back and I've realised there is a bit I've missed, it'll get deleted before the new code is worked on. – [Colin Mackay](#) Jul 29, 2009 at 19:18

-
- 4 I say check in the commented code once, THEN delete it. There are many times when you test various different bits of

code, and you don't want to check in broken code...

– [DisgruntledGoat](#) Jul 31, 2009 at 13:33

Not to mention that version control is your friend.

– [David Thornley](#) Feb 1, 2010 at 19:12

+1 I worked with a programmer that insisted on commenting **all** of the code that he had refactored or rewritten. It would drive me crazy because sometimes I would have to scroll through 1k+ lines of crap to find what I was working on. – [Evan Plaise](#) Jun 25, 2010 at 5:24

45

votes



The overuse / abuse of `#region` directives. It's just a little thing, but in C#, I previously would use `#region` directives all over the place, to organize my classes. For example, I'd group all class properties together in a region.

Now I look back at old code and mostly just get annoyed by them. I don't think it really makes things clearer most of the time, and sometimes they just plain slow you down. So I have now changed my mind and feel that well laid out classes are mostly cleaner *without* region directives.

Share

[edited Jul 7, 2009 at 3:25](#)

community wiki

[2 revs](#)

[Scott Ferguson](#)

31 I hate region's. People on my team use them frivolously. I call them "bad code hidlers". – [rball](#) Jul 6, 2009 at 22:01

9 They're definitely a code smell. – [Frank Schwieterman](#) Jul 6, 2009 at 22:08

3 I HATE regions. I am currently maintaining code where function is almost 500 lines and to manage it, the smart developer has put chunks of code in 10 to 15 regions. – [SolutionYogi](#) Jul 7, 2009 at 2:28

6 @Solution Yogi: I don't think regions are the real problem in your case :-) – [Ed Swangren](#) Jul 7, 2009 at 2:39

9 I think regions can be fine if used sparingly. – [Gregory Higley](#) Jul 7, 2009 at 10:13

38
votes



Waterfall development in general, and in specific, the practice of writing complete and comprehensive functional and design specifications that are somehow expected to be canonical and then expecting an implementation of those to be correct and acceptable. I've seen it replaced with Scrum, and good riddance to it, I say. The simple fact is that the changing nature of customer needs and desires makes any fixed specification effectively useless; the only way to really properly approach the problem is with an iterative approach. Not that Scrum is a silver bullet, of course; I've seen it misused and abused many, many times. But it beats waterfall.

Share

answered [Jul 6, 2009 at 22:30](#)

community wiki
[Paul Sonier](#)

-
- 3 Tell that to my customer... I'm in the middle of writing some useless "I'm a programmer with a crystal ball so I know exactly how my low-level design will look like in 6 months" specification document :) – [Igor Brejc](#) Jul 27, 2009 at 13:44
-

35 Never crashing.

votes



It seems like such a good idea, doesn't it? Users don't like programs that crash, so let's write programs that don't crash, and users should like the program, right? That's how I started out.

Nowadays, I'm more inclined to think that if it doesn't work, it shouldn't pretend it's working. Fail as soon as you can, with a good error message. If you don't, your program is going to crash even harder just a few instructions later, but with some nondescript null-pointer error that'll take you an hour to debug.

My favorite "don't crash" pattern is this:

```
public User readUserFromDb(int id){
    User u = null;
    try {
        ResultSet rs = connection.execute("SELECT *
FROM user WHERE id = " + id);
        if (rs.moveToNext()){
            u = new User();
            u.setFirstName(rs.get("fname"));
            u.setSurname(rs.get("sname"));
            // etc
        }
    } catch (Exception e) {
        log.info(e);
    }
}
```

```
    }
    if (u == null){
        u = new User();
        u.setFirstName("error communicating with
database");
        u.setSurname("error communicating with
database");
        // etc
    }
    u.setId(id);
    return u;
}
```

Now, instead of asking your users to copy/paste the error message and sending it to you, you'll have to dive into the logs trying to find the log entry. (And since they entered an invalid user ID, there'll be no log entry.)

Share

edited Jul 8, 2009 at 6:27

community wiki
[2 revs](#)
[gustafc](#)

What's the likelihood of the user giving you the actual error message, vs your logs producing the issue? (Very low in this particular case, but users almost never quote the error messages!) Do they even read them? – [Arafangion](#) Jul 7, 2009 at 10:46

- 1 I admit the chance is low that a random user sends you the error message, but the chance is non-zero (trivial example: sometimes you use your own app), and some users actually learn with time what to copy/paste. I'm not saying you shouldn't log (you should), but when the app is broken, it **is** broken.

Showing an error message is far better, far more honest to the user than pretending that the user's first name is "error communicating with database" (or even worse, `null` or the empty string). – [gustafc](#) Jul 7, 2009 at 11:13

There's an `NullPointerException` on line two – [oɔwəɹ](#) Jul 7, 2009 at 20:38

Thanks, oɔwəɹ, I fixed it. (Although it was a bit lulzier with it there: All this trouble to avoid exceptions and other "crashes", and still it unconditionally crashed.) – [gustafc](#) Jul 8, 2009 at 6:29

33

votes



I thought it made sense to apply **design patterns** whenever I recognised them.

Little did I know that I was actually copying styles from foreign programming languages, while the language I was working with allowed for far more elegant or easier solutions.

Using multiple (very) different languages opened my eyes and made me realise that I don't have to mis-apply other people's solutions to problems that aren't mine. Now I shudder when I see the *factory pattern* applied in a language like Ruby.

Share

answered [Jul 6, 2009 at 21:52](#)

community wiki
[molf](#)

2 Please excuse my ignorance of ruby, but why should we not use the factory pattern with it? – [Mike Chamberlain](#) Sep 6, 2011 at 23:50

Not a rubyist here, but the factory is to avoid depending on an implementation, but ruby is dynamic, and you can mock or stub Anything. So you don't really depend on an implementation.
– [Stéphane](#) Sep 20, 2011 at 20:34

27
votes



Obsessive testing. I used to be a rabid proponent of test-first development. For some projects it makes a lot of sense, but I've come to realize that it is not only unfeasible, but rather detrimental to many projects to slavishly adhere to a doctrine of writing unit tests for every single piece of functionality.

Really, slavishly adhering to *anything* can be detrimental.

Share

answered [Jul 6, 2009 at 22:01](#)

community wiki
[yalestar](#)

22 It works out pretty well for barnacles. – [MusiGenesis](#) Jul 7, 2009 at 0:55

Test coverage has to be proportional to the benefit. Anything you do really has to show a benefit. 100% coverage isn't going to give you all that much. difference from 80 or 90 in a form that isn't in a life support / missile launch scenario. – [Spence](#)
Jul 7, 2009 at 3:34

+1 reliance on unit testing as opposed to testing.

– [Preet Sangha](#) Jul 11, 2009 at 10:50

25
votes



This is a small thing, but: Caring about where the braces go (on the same line or next line?), suggested maximum line lengths of code, naming conventions for variables, and other elements of style. I've found that everyone seems to care more about this than I do, so I just go with the flow of whoever I'm working with nowadays.

Edit: The exception to this being, of course, when I'm the one who cares the most (or is the one in a position to set the style for a group). In that case, I do what I want!

(Note that this is not the same as having no consistent style. I think a consistent style in a codebase is very important for readability.)

Share

[edited Jul 6, 2009 at 22:00](#)

community wiki

[2 revs](#)

[Daniel Lew](#)

5 Someone gave this a downvote, but I think its a practical perspective. What is the best code styling? Not important. Look up and down in the same file and duplicate.

– [Frank Schwieterman](#) Jul 6, 2009 at 21:55

- 12 The best code styling is whatever the standard is for that shop.
– [David Thornley](#) Jul 6, 2009 at 22:00

Thats why I love the auto format options in Visual Studio. It does not matter how the other developers wrote the code I just do a quick format and its exactly how I like it... most the time.

– [corymathews](#) Jul 7, 2009 at 3:34

-
- 5 @cory: doesn't that mess up the ability of your version control software to show you the difference between versions of the file you're just reformatted? – [Steve Melnikoff](#) Jul 7, 2009 at 20:29

Which is why I'm kind of attracted to learning python... to think I just have to worry about what my tabstops are set to, and not bracing styles. It's kind of compelling. – [Chris K](#) Jul 20, 2009 at 14:23

24
votes



Perhaps the most important "programming practice" I have since changed my mind about, is the idea that my code is better than everyone else's. This is common for programmers (especially newbies).

Share

answered [Jul 7, 2009 at 2:37](#)

community wiki
[Nippysaurus](#)

20
votes



Utility libraries. I used to carry around an assembly with a variety of helper methods and classes with the theory that I could use them somewhere else someday.



In reality, I just created a huge namespace with a lot of poorly organized bits of functionality.

Now, I just leave them in the project I created them in. In all probability I'm not going to need it, and if I do, I can always refactor them into something reusable later. Sometimes I will flag them with a `//TODO` for possible extraction into a common assembly.

Share

answered [Jul 6, 2009 at 22:45](#)

community wiki
[JamesWampler](#)

12 There's a good quote (I can't find the original at the moment) which was something along the lines of "don't even think about creating a generic routine until you've needed to solve the same problem 3 times. – [DaveR](#) Jul 6, 2009 at 23:34

9 "Three strikes and you refactor" - *Refactoring* by Martin Fowler. **The Rule of Three**, pg 58. – [Nick Dandoulakis](#) Jul 7, 2009 at 5:00

20
votes

Designing more than I coded. After a while, it turns into analysis paralysis.



Share

answered [Jul 6, 2009 at 22:59](#)



community wiki



44 I occasionally invoke the phrase "If you find that you are thinking too much, stop and do. If you find that you are doing too much, stop and think." – [Neil N](#) Jul 6, 2009 at 23:02

That is nice, but how much is too much? – [Hamish Grubijan](#)
Jun 3, 2010 at 21:39

Too much dependence on UML (Useless Modeling Language). It **occasionally** has its uses. But once I see someone start to draw class diagrams and preach to the benefits of "how awesome it would be to generate code from the diagrams" I lace up my running shoes. Plus, Visual Studio has a build-in interactive class diagram generator that does it all automatically and works like the object explorer on crack.
– [Evan Plai](#) Jun 25, 2010 at 6:50

15
votes

The use of a DataSet to perform business logic. This binds the code too tightly to the database, also the DataSet is usually created from SQL which makes things even more fragile. If the SQL or the Database changes it tends to trickle to everything the DataSet touches.

Performing any business logic inside an object constructor. With inheritance and the ability to create overloaded constructors tend to make maintenance difficult.

Share

answered [Jul 7, 2009 at 2:24](#)

community wiki
[user117499](#)

15 Abbreviating variable/method/table/... Names

votes



I used to do this all of the time, even when working in languages with no enforced limits on lengths of names (well they were probably 255 or something). One of the side-effects were a lot of comments littered throughout the code explaining the (non-standard) abbreviations. And of course, if the names were changed for any reason...

Now I much prefer to call things what they really are, with good descriptive names. including *standard* abbreviations only. No need to include useless comments, and the code is far more readable and understandable.

Share

answered [Jul 7, 2009 at 9:59](#)

community wiki
[Rhys Jones](#)

Yes, gotta love these types of declarations: void Foo(x1,y,x2,y2,p,r,j)...WTF?! – [Ed Swangren](#) [Jul 11, 2009 at 19:07](#)

Or worse (and yes, I've actually seen this), `Foo(int arg0, String arg1, float arg2)` etc – [Mac](#) Oct 5, 2011 at 3:08

14 votes Wrapping existing Data Access components, like the Enterprise Library, with a custom layer of helper methods.



- It doesn't make anybody's life easier
- Its more code that can have bugs in it
- A lot of people know how to use the EntLib data access components. No one but the local team knows how to use the in house data access solution

Share

answered [Jul 6, 2009 at 21:53](#)

community wiki
[blu](#)

14 votes I first heard about object-oriented programming while reading about Smalltalk in 1984, but I didn't have access to an o-o language until I used the cfront C++ compiler in 1992. I finally got to use Smalltalk in 1995. I had eagerly anticipated o-o technology, and bought into the idea that it would save software development.

Now, I just see o-o as one technique that has some advantages, but it's just one tool in the toolbox. I do most of my work in Python, and I often write standalone functions

that are not class members, and I often collect groups of data in tuples or lists where in the past I would have created a class. I still create classes when the data structure is complicated, or I need behavior associated with the data, but I tend to resist it.

I'm actually interested in doing some work in Clojure when I get the time, which doesn't provide o-o facilities, although it can use Java objects if I understand correctly. I'm not ready to say anything like o-o is dead, but personally I'm not the fan I used to be.

Share

answered [Jul 7, 2009 at 3:10](#)

community wiki
[Greg Graham](#)

13 In C#, using `_notation` for private members. I now think it's ugly.

votes



I then changed to `this.notation` for private members, but found I was inconsistent in using it, so I dropped that too.

Share


answered [Jul 6, 2009 at 22:13](#)


community wiki
[JulianR](#)

29 I'm still using _notation and think it's great. – [Arnis Lapsa](#) Jul 8, 2009 at 9:24

3 I hate _notation; I use ThisNotation for public members and thisNotation for private members. – [Callum Rogers](#) Jul 20, 2009 at 16:17

I hate it too. It confuses me :(– [Broken_Window](#) Aug 4, 2009 at 4:14

4 I disagree. It makes it so much easier to manage names. Use PascalCase for properties or public/internal members, _UnderscorePascalCase for members that are exposed through a property, and camelCase for parameter names in methods/constructors and private members. The 'this' keyword is only necessary if you need to pass the reference of the current class outside of the class or you need to access an auto-generated member within the class (such as name, controls, etc...). – [Evan Plaice](#) Jun 25, 2010 at 7:03 

@Evan: I do exactly what you're describing except for the last part. I tend to use the `this` or `Me` (C# & VB.NET respectively) when calling methods and properties. IMO, it makes my code easier to read and understand later, especially when there are four or more objects within that particular scope. – [Alex Essilfie](#) Aug 18, 2011 at 20:24 

11
votes



I stopped going by the university recommended method of design before implementation. Working in a chaotic and complex system has forced me to change attitude.

Of course I still do code research, especially when I'm about to touch code I've never touched before, but normally I try to focus on as small implementations as possible to get something going first. This is the primary goal. Then

gradually refine the logic and let the design just appear by itself. Programming is an **iterative process** and works very well with an agile approach and with lots of refactoring.

The code will not look at all what you first thought it would look like. Happens every time :)

Share

answered [Jul 6, 2009 at 22:34](#)

community wiki
[ralphtheninja](#)

10

votes



I used to be big into design-by-contract. This meant putting a lot of error checking at the beginning of all my functions. Contracts are still important, from the perspective of separation of concerns, but rather than try to enforce what my code shouldn't do, I try to use unit tests to verify what it does do.

Share

answered [Jul 6, 2009 at 21:53](#)

community wiki

[Frank Schwieterman](#)

I was taught to program like this. Of course, I was being taught by a HS math teacher, so I suppose it makes sense that he wanted his functions all self-verifying. – [Alex](#) [Jul 6, 2009 at 21:57](#)

10

votes



I would use static's in a lot of methods/classes as it was more concise. When I started writing tests that practice changed very quickly.

Share

answered [Jul 6, 2009 at 22:00](#)

community wiki

[rball](#)

10 Checked Exceptions

votes



An amazing idea on paper - defines the contract clearly, no room for mistake or forgetting to check for some exception condition. I was sold when I first heard about it.

Of course, it turned to be such a mess in practice. To the point of having libraries today like Spring JDBC, which has hiding legacy checked exceptions as one of its main features.

Share

answered [Jul 23, 2009 at 19:00](#)

community wiki
[Gregory Mostizky](#)

9

votes



That anything worthwhile was only coded in one particular language. In my case I believed that C was the best language ever and I never had any reason to code anything in any other language... ever.

I have since come to appreciate many different languages and the benefits/functionality they offer. If I want to code something small - quickly - I would use Python. If I want to work on a large project I would code in C++ or C#. If I want to develop a brain tumour I would code in [Perl](#).

Share

edited [May 23, 2017 at 12:07](#)

community wiki

[2 revs](#)

[Patrick Gryciuk](#)

8

votes



When I needed to do some refactoring, I thought it was faster and cleaner to start straightaway and implement the new design, fixing up the connections until they work. Then I realized it's better to do a series of small refactorings to slowly but reliably progress towards the new design.

Share

answered [Jul 7, 2009 at 2:59](#)

community wiki

[Ying](#)

can remember the number of times this has bit me....

– [Preet Sangha](#) [Jul 11, 2009 at 10:46](#)

8

votes



Perhaps the biggest thing that has changed in my coding practices, as well as in others, is the acceptance of outside classes and libraries downloaded from the internet as the basis for behaviors and functionality in applications. In school at the time I attended college we were encouraged to figure out how to make things better via our own code and rely upon the language to solve our problems. With the advances in all aspects of user interface and service/data consumption this is no longer a realistic notion.

There are certain things which will never change in a language, and having a library that wraps this code in a simpler transaction and in fewer lines of code that I have to write is a blessing. Connecting to a database will always be the same. Selecting an element within the DOM will not change. Sending an email via a server-side script will never change. Having to write this time and again wastes time that I could be using to improve my core logic in the application.

Share

answered [Jul 7, 2009 at 10:24](#)

community wiki
[Liam](#)

7 Initializing all class members.

votes





I used to explicitly initialize every class member with something, usually NULL. I have come to realize that this:

- normally means that every variable is initialized twice before ever being read
- is silly because in most languages automatically initialize variables to NULL.
- actually enforces a slight performance hit in most languages
- can bloat code on larger projects

community wiki
[Nippysaurus](#)

-
- 4 Sometimes the consequences of NOT initializing all class members can really bite you in the a\$\$ though. – [muusbolla](#) Jul 10, 2009 at 17:46
-

Unless you're using prototype based language that creates new instances by cloning. Initializing all members can really save you a lot of trouble. – [Wojciech Bederski](#) Jul 24, 2009 at 11:06

6
votes



Like you, I also have embraced IoC patterns in reducing coupling between various components of my apps. It makes maintenance and parts-swapping much simpler, as long as I can keep each component as independent as possible. I'm also utilizing more object-relational frameworks such as NHibernate to simplify database management chores.

In a nutshell, I'm using "mini" frameworks to aid in building software more quickly and efficiently. These mini-frameworks save lots of time, and if done right can make an application super simple to maintain down the road. Plug 'n Play for the win!

-1 I can't stand the proliferation of IoC and frameworks.
Decoupling good, IoC and frameworks = needless complexity
– [Paul Hollingsworth](#) Jul 7, 2009 at 10:39

How can you praise decoupling and yet still hate on IoC and other frameworks? That's what many IoC frameworks and design patterns do to begin with. – [ajawad987](#) Jul 8, 2009 at 6:17

1

2

Next