What's your most controversial programming opinion?

Asked 15 years, 11 months ago Modified 1 year, 11 months ago

Viewed 312k times

363

votes

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.

This is definitely subjective, but I'd like to try to avoid it becoming argumentative. I think it could be an interesting question if people treat it appropriately.

The idea for this question came from the comment thread from my answer to the "What are five things you hate about your favorite language?" question. I contended that classes in C# should be sealed by default - I won't put my reasoning in the question, but I might write a fuller explanation as an answer to this question. I was surprised at the heat of the discussion in the comments (25 comments currently).

So, what contentious opinions do *you* hold? I'd rather avoid the kind of thing which ends up being pretty religious with relatively little basis (e.g. brace placing) but examples might include things like "unit testing isn't actually terribly helpful" or "public fields are okay really". The important thing (to me, anyway) is that you've got reasons behind your opinions.

Please present your opinion and reasoning - I would encourage people to vote for opinions which are well-argued and interesting, whether or not you happen to agree with them.

language-agnostic

Share

edited May 23, 2017 at 12:10

community wiki
6 revs, 4 users 61%
Jon Skeet

Comments disabled on deleted / locked posts / reviews

407 Answers

Sorted by: Highest score (default)

1 2 3 4 5 … 14 Next

873
votes

Programmers who don't code in their spare time for fun will never become as good as those that do.

I think even the smartest and most talented people will never become truly good programmers unless they treat it as more than a job. Meaning that they do little projects on the side, or just mess with lots of different languages and ideas in their spare time.

(Note: I'm not saying good programmers do nothing else than programming, but they do more than program from 9 to 5)

Share

edited Apr 17, 2010 at 22:34

community wiki
3 revs, 3 users 57%
rustyshelf

767
votes

The only "best practice" you should be using all the time is "Use Your Brain".

Too many people jumping on too many bandwagons and trying to force methods, patterns, frameworks etc onto things that don't warrant them. Just because something is new, or because someone respected has an opinion, doesn't mean it fits all :)

EDIT: Just to clarify - I don't think people should ignore best practices, valued opinions etc. Just that people shouldn't just blindly jump on something without thinking about WHY this "thing" is so great, IS it applicable to what I'm doing, and WHAT benefits/drawbacks does it bring?

Share

edited Jan 9, 2009 at 15:52

community wiki
4 revs, 2 users 90%
Steven Robbins

710
votes

Most comments in code are in fact a pernicious form of code duplication.

We spend most of our time maintaining code written by others (or ourselves) and poor, incorrect, outdated, misleading comments must be near the top of the list of most annoying artifacts in code.

I think eventually many people just blank them out, especially those flowerbox monstrosities.

Much better to concentrate on making the code readable, refactoring as necessary, and minimising idioms and quirkiness.

On the other hand, many courses teach that comments are very nearly more important than the code itself, leading to the *this next line adds one to invoiceTotal* style of commenting.

Share

edited Apr 11, 2009 at 20:13

community wiki
6 revs, 2 users 95%
Ed Guiness

709
votes

"Googling it" is okay!

Yes, I know it offends some people out there that their years of intense memorization and/or glorious stacks of programming books are starting to fall by the wayside to a resource that anyone can access within seconds, but you shouldn't hold that against people that use it.

Too often I hear googling answers to problems the result of criticism, and it really is without sense. First of all, it must be conceded that everyone needs materials to reference. You don't know everything and you will need to look things up. Conceding that, does it really matter where you got the information? Does it matter if you looked it up in a book, looked it up on Google, or heard it from a talking frog that you hallucinated? No. A right answer is a right answer.

What is important is that you understand the material, use it as the means to an end of a successful programming solution, and the client/your employer is happy with the results.

(although if you are getting answers from hallucinatory talking frogs, you should probably get some help all the same)

Share

edited Jan 2, 2009 at 21:12

community wiki
2 revs
PhoenixRedeemer

692 **XML is highly overrated**

votes

I think too many jump onto the XML bandwagon before using their brains... XML for web stuff is great, as it's designed for it. Otherwise I think some *problem definition* and *design* thoughts should preempt any decision to use it.

My 5 cents

Share

answered Jan 3, 2009 at 11:04

community wiki
Over Rated

678

votes

Not all programmers are created equal

Quite often managers think that DeveloperA == DeveloperB simply because they have same level of experience and so on. In actual fact, the performance of one developer can be 10x or even 100x that of another.

It's politically risky to talk about it, but sometimes I feel like pointing out that, even though several team members may *appear* to be of equal skill, it's not always the case. I have even seen cases where lead developers were 'beyond hope' and junior devs did all the actual work - I made sure they got the credit, though. :)

Share

edited Jan 27, 2009 at 9:50

community wiki
5 revs, 3 users 82%
Dmitri Nesteruk

612
votes

I fail to understand why people think that Java is absolutely the best "first" programming language to be taught in universities.

For one, I believe that first programming language should be such that it highlights the need to learn control flow and variables, not objects and syntax

For another, I believe that people who have not had experience in debugging memory leaks in C / C++ cannot fully appreciate what Java brings to the table.

Also the natural progression should be from "how can I do this" to "how can I find the library which does that" and not the other way round.

Share

edited Jan 12, 2009 at 17:41

community wiki
6 revs, 4 users 75%
Learning

538
votes

If you only know one language, no matter how well you know it, you're not a great programmer.

There seems to be an attitude that says once you're really good at C# or Java or whatever other language you started out learning then that's all you need. I don't believe it- every language I have ever learned has taught me something new about programming that I have been able to bring back into my work with all the others. I think that anyone who restricts themselves to one language will never be as good as they could be.

It also indicates to me a certain lack of inquistiveness and willingness to experiment that doesn't necessarily tally with the qualities I would expect to find in a really good programmer.

Share

answered Jan 2, 2009 at 13:59

community wiki
glenatron

534
votes

Performance *does* matter.

Share

edited Jan 9, 2009 at 16:19

community wiki
2 revs, 2 users 80%
Daniel Paull

488

votes

Print statements are a valid way to debug code

I believe it is perfectly fine to debug your code by littering it with `System.out.println` (or whatever print statement works for your language). Often, this can be quicker than debugging, and you can compare printed outputs against other runs of the app.

Just make sure to remove the print statements when you go to production (or better, turn them into logging statements)

Share

edited Jan 2, 2009 at 20:45

community wiki
2 revs
David

466

votes

Your job is to put yourself out of work.

When you're writing software for your employer, any software that you create is to be written in such a way that it can be picked up by any developer and understood with a minimal amount of effort. It is well designed, clearly and consistently written, formatted cleanly, documented where it needs to be, builds daily as expected, checked into the repository, and appropriately versioned.

If you get hit by a bus, laid off, fired, or walk off the job, your employer should be able to replace you on a moment's notice, and the next guy could step into your role, pick up your code and be up and running within a week tops. If he or she can't do that, then you've failed miserably.

Interestingly, I've found that having that goal has made me more valuable to my employers. The more I strive to be disposable, the more valuable I become to them.

Share

answered Jan 4, 2009 at 10:33

community wiki
Mike Hofer

464
votes

1) The Business Apps farce:

I think that the whole "Enterprise" frameworks thing is smoke and mirrors. J2EE, .NET, the majority of the Apache frameworks and most abstractions to manage such things create far more complexity than they solve.

Take any regular Java or .NET ORM, or any supposedly modern MVC framework for either which does "magic" to solve tedious, simple tasks. You end up writing huge amounts of ugly XML boilerplate that is difficult to validate and write quickly. You have massive APIs where half of those are just to integrate the work of the other APIs, interfaces that are impossible to recycle, and abstract classes that are needed only to overcome the inflexibility of Java and C#. We simply don't need most of that.

How about all the different application servers with their own darned descriptor syntax, the overly complex database and groupware products?

The point of this is not that complexity==bad, it's that unnecessary complexity==bad. I've worked in massive enterprise installations where some of it was necessary, but even in most cases a few home-grown scripts and a simple web frontend is all that's needed to solve most use cases.

I'd try to replace all of these enterprisey apps with simple web frameworks, open source DBs, and trivial programming constructs.

2) The n-years-of-experience-required:

Unless you need a consultant or a technician to handle a specific issue related to an application, API or framework, then you don't really need someone with 5 years of experience in that application. What you need is a developer/admin who can read documentation, who has domain knowledge in whatever it is you're doing, and who can learn quickly. If you need to develop in some kind of language, a decent developer will pick it up in less than 2 months. If you need an administrator for X web server, in two days he should have read the man pages and newsgroups and be up to speed. Anything less and that person is not worth what he is paid.

3) The common "computer science" degree curriculum:

The majority of computer science and software engineering degrees are bull. If your first programming language is Java or C#, then you're doing something wrong. If you don't get several courses full of algebra and math, it's wrong. If you don't delve into functional programming, it's incomplete. If you can't apply loop invariants to a trivial for loop, you're not worth your salt as a supposed computer scientist. If you come out with experience in x and y languages and object orientation, it's full of s***. A real computer scientist sees a language in terms of the concepts and syntaxes it uses, and sees programming methodologies as one among many, and has such a good understanding of the underlying philosophies of both that picking new languages, design methods, or specification languages should be trivial.

Share

edited Oct 25, 2009 at 22:22

community wiki
5 revs, 5 users 77%
Daishiman

438
votes

Getters and Setters are Highly Overused

I've seen millions of people claiming that public fields are evil, so they make them private and provide getters and setters for all of them. I believe this is almost identical to making the fields public, maybe a bit different if you're using threads (but generally is not the case) or if your accessors have business/presentation logic (something 'strange' at least).

I'm not in favor of public fields, but against making a getter/setter (or Property) for everyone of them, and then claiming that doing that is **encapsulation** or **information hiding**... ha!

UPDATE:

This answer has raised some controversy in it's comments, so I'll try to clarify it a bit (I'll leave the original untouched since that is what many people upvoted).

First of all: **anyone who uses public fields deserves jail time**

Now, creating private fields and then using the IDE to *automatically generate getters and setters for every one of them* is **nearly as bad** as using public fields.

Many people think:

```
private fields + public accessors == encapsulation
```

I say (automatic or not) generation of getter/setter pair for your fields effectively goes against the so called encapsulation you are trying to achieve.

Lastly, let me quote Uncle Bob in this topic (taken from chapter 6 of "Clean Code"):

> There is a reason that we keep our variables private. We don't want anyone else to depend on them. We want the freedom to change their type or implementation on a whim or an impulse. Why, then, do so many programmers automatically add getters and setters to their objects, exposing their private fields **as if they were public?**

Share

edited May 15, 2009 at 16:25

community wiki
4 revs, 2 users 98%
Pablo Fernandez

383
votes

UML diagrams are highly overrated

Of course there are useful diagrams e.g. class diagram for the Composite Pattern, but many UML diagrams have absolutely no value.

Share

edited Jan 8, 2009 at 12:21

community wiki
2 revs, 2 users 83%
Ludwig Wensauer

380 votes

Opinion: SQL is code. Treat it as such

That is, just like your C#, Java, or other favorite object/procedure language, develop a formatting style that is readable and maintainable.

I hate when I see sloppy free-formatted SQL code. If you scream when you see both styles of curly braces on a page, why or why don't you scream when you see free formatted SQL or SQL that obscures or obfuscates the JOIN condition?

Share

edited Aug 17, 2010 at 15:45

community wiki
2 revs, 2 users 92%
MustStayAnonymous

354 votes

Readability is the most important aspect of your code.

Even more so than correctness. If it's readable, it's easy to fix. It's also easy to optimize, easy to change, easy to understand. And hopefully other developers can learn something from it too.

Share

answered Jan 2, 2009 at 16:40

community wiki
Craig P. Motlin

---

341 votes

If you're a developer, you should be able to write code

I did quite a bit of interviewing last year, and for my part of the interview I was supposed to test the way people thought, and how they implemented simple-to-moderate algorithms on a white board. I'd initially started out with questions like:

> Given that Pi can be estimated using the function 4 * (1 - 1/3 + 1/5 - 1/7 + ...) with more terms giving greater accuracy, write a function that calculates Pi to an accuracy of 5 decimal places.

It's a problem that should make you think, but shouldn't be out of reach to a seasoned developer (it can be answered in about 10 lines of C#). However, many of our (supposedly pre-screened by the agency) candidates couldn't even begin to answer it, or even explain how they might go about answering it. So after a while I started asking simpler questions like:

> Given the area of a circle is given by Pi times the radius squared, write a function to calculate the area of a circle.

Amazingly, more than half the candidates couldn't write this function in *any* language (I can read most popular languages so I let them use any language of their choice, including pseudo-code). We had "C# developers" who could not write this function in C#.

I was surprised by this. I had always thought that developers should be able to write code. It seems that, nowadays, this is a controversial opinion. Certainly it is amongst interview candidates!

Edit:

There's a lot of discussion in the comments about whether the first question is a good or bad one, and whether you should ask questions as complex as this in an interview. I'm not going to delve into this here (that's a whole new question) apart from to say *you're largely missing the point of the post*.

Yes, I said people couldn't make any headway with this, but the second question is *trivial* and many people couldn't make any headway with that one either! *Anybody* who calls themselves a developer should be able to write the answer to the second one in a few seconds without even thinking. And many can't.

Share

edited Jan 8, 2009 at 23:16

community wiki

3 revs
Greg Beech

330
votes

The use of hungarian notation should be punished with death.

That should be controversial enough ;)

Share

edited Jan 9, 2009 at 16:30

community wiki
2 revs, 2 users 80%
Marc

287
votes

Design patterns are hurting good design more than they're helping it.

IMO software design, especially good software design is far too varied to be meaningfully captured in patterns, especially in the small number of patterns people can actually remember - and they're far too abstract for people to really remember more than a handful. So they're not helping much.

And on the other hand, far too many people become enamoured with the concept and try to apply patterns everywhere - usually, in the resulting code you can't find the actual design between all the (completely meaningless) Singletons and Abstract Factories.

Share

edited Jan 2, 2009 at 14:30

community wiki
3 revs, 2 users 90%
Michael Borgwardt

---

274 votes

Less code is better than more!

If the users say "that's it?", and your work remains invisible, it's done right. Glory can be found elsewhere.

Share

answered Jan 4, 2009 at 3:32

community wiki
Jas Panesar

---

265 votes

PHP sucks ;-)

The proof is in the pudding.

Share

answered Jan 2, 2009 at 18:00

community wiki
Doubting Thomas

261

Unit Testing won't help you write good code

votes

The only reason to have Unit tests is to make sure that code that already works doesn't break. Writing tests first, or writing code to the tests is ridiculous. If you write to the tests before the code, you won't even know what the edge cases are. You could have code that passes the tests but still fails in unforeseen circumstances.

And furthermore, good developers will keep cohesion low, which will make the addition of new code unlikely to cause problems with existing stuff.

In fact, I'll generalize that even further,

Most "Best Practices" in Software Engineering are there to keep bad programmers from doing too much damage.

They're there to hand-hold bad developers and keep them from making dumbass mistakes. Of course, since most developers are bad, this is a good thing, but good developers should get a pass.

Share

answered Jan 2, 2009 at 15:08

community wiki
Chad Okere

256
votes

Write small methods. It seems that programmers love to write loooong methods where they do multiple different things.

I think that a method should be created wherever you can name one.

Share

edited Jan 9, 2009 at 16:29

community wiki
2 revs, 2 users 80%
Matt Secoske

---

234 votes

It's ok to write garbage code once in a while

Sometimes a quick and dirty piece of garbage code is all that is needed to fulfill a particular task. Patterns, ORMs, SRP, whatever... Throw up a Console or Web App, write some inline sql (feels good), and blast out the requirement.

Share

answered Jan 2, 2009 at 14:21

community wiki
John Farrell

---

194 votes

Code == Design

I'm no fan of sophisticated UML diagrams and endless code documentation. In a high level language, your code should be readable and understandable as is. Complex documentation and diagrams aren't really any more user friendly.

Here's an article on the topic of Code as Design.

Share

edited Jan 23, 2009 at 4:49

community wiki
2 revs
Jon B

186
votes

Software development is just a job

Don't get me wrong, I enjoy software development a lot. I've written a blog for the last few years on the subject. I've spent enough time on here to have >5000 reputation points. And I work in a start-up doing typically 60 hour weeks for much less money than I could get as a contractor because the team is fantastic and the work is interesting.

But in the grand scheme of things, it is just a job.

It ranks in importance below many things such as family, my girlfriend, friends, happiness etc., and below other things I'd rather be doing if I had an unlimited supply of cash such as riding motorbikes, sailing yachts, or snowboarding.

I think sometimes a lot of developers forget that developing is just something that allows us to have the more important things in life (and to have them by doing something we enjoy) rather than being the end goal in itself.

Share

answered Jan 2, 2009 at 15:17

community wiki
Greg Beech

185

votes

I also think there's **nothing wrong with having binaries in source control**.. if there is a good reason for it. If I have an assembly I don't have the source for, and might not necessarily be in the same place on each devs machine, then I will usually stick it in a "binaries" directory and reference it in a project using a relative path.

Quite a lot of people seem to think I should be burned at the stake for even mentioning "source control" and "binary" in the same sentence. I even know of places that have strict rules saying you can't add them.

Share

edited Jan 9, 2009 at 16:36

community wiki

2 revs, 2 users 80%
Steven Robbins

179

votes

Every developer should be familiar with the basic architecture of modern computers. This also applies to developers who target a virtual machine (maybe even more so, because they have been told time and time again that they don't need to worry themselves with memory management etc.)

Share

edited Jan 3, 2009 at 21:39

community wiki
2 revs
Brian Rasmussen

162
votes

Software Architects/Designers are Overrated

As a developer, I hate the idea of Software Architects. They are basically people that no longer code full time, read magazines and articles, and then tell you how to design software. Only people that actually write software full time for a living should be doing that. I don't care if you were the worlds best coder 5 years ago before you became an Architect, your opinion is useless to me.

How's that for controversial?

Edit (to clarify): I think most Software Architects make great Business Analysts (talking with customers, writing requirements, tests, etc), I simply think they have no place in designing software, high level or otherwise.

Share

edited Jan 4, 2009 at 5:45

community wiki
2 revs
rustyshelf

151
votes

There is no "one size fits all" approach to development

I'm surprised that this is a controversial opinion, because it seems to me like common sense. However, there are many entries on popular blogs promoting the "one size fits all" approach to development so I think I may actually be in the minority.

Things I've seen being touted as *the* correct approach for *any* project - before any information is known about it - are things like the use of Test Driven Development (TDD), Domain Driven Design (DDD), Object-Relational Mapping (ORM), Agile (capital A), Object Orientation (OO), etc. etc. encompassing everything from methodologies to architectures to components. All with nice marketable acronyms, of course.

People even seem to go as far as putting badges on their blogs such as "I'm Test Driven" or similar, as if their strict adherence to a single approach whatever the details of the project project is actually a *good thing*.

It isn't.

Choosing the correct methodologies and architectures and components, etc., is something that should be done on a *per-project* basis, and depends not only on the type of project you're working on and its unique requirements, but also the size and ability of the team you're working with.

Share answered Jan 2, 2009 at 14:21

community wiki
Greg Beech