What OOP coding practices should you always make time for?

Asked 16 years, 1 month ago Modified 15 years, 8 months ago

Viewed 1k times

13

I tend to do a lot of projects on short deadlines and with lots of code that will never be used again, so there's always pressure/temptation to cut corners. One rule I always stick to is encapsulation/loose coupling, so I have lots of small classes rather than one giant God class. But what else should I never compromise on?

Update - thanks for the great response. Lots of people have suggested unit testing, but I don't think that's really appropriate to the kind of UI coding I do. Usability / User acceptance testing seems much important. To reiterate, I'm talking about the BARE MINIMUM of coding standards for impossible deadline projects.

oop rad

Share

Improve this question

Follow

edited Apr 8, 2009 at 20:46

George Stocker

57.9k ● 29 ● 180 ● 238

asked Nov 24, 2008 at 14:18

Iain

9,442 ● 11 ● 50 ● 65

19 Answers

Sorted by: Highest score (default)

19

Not OOP, but a practice that helps in both the short and long run is DRY, Don't Repeat Yourself. Don't use copy/paste inheritance.

Share Improve this answer Follow

answered Nov 24, 2008 at 14:19
Corey Trager
23.1k ● 20 ● 87 ● 121

LOl, I just removed some duplicate code despite the deadline ;-). – Toon Krijthe Nov 24, 2008 at 14:22

1 I've never heard it called "copy/paste inheritance" before... but it fits perfectly. – Dave Sherohman Nov 24, 2008 at 16:51

13

Not a OOP practice, but common sense ;-).

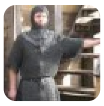
If you are in a hurry, and have to write a hack. Always add a piece of comment with the reasons. So you can trace it back and make a good solution later.

If you never had the time to come back, you always have the comment so you know, why the solution was chosen at the moment.

Share Improve this answer

Follow

answered Nov 24, 2008 at 14:21



Toon Krijthe

53.4k ● 38 ● 149 ● 202

2 And throw some kind of flag in there so you can easily find such comments. I just use TODO, because Eclipse will flag that with a blue mark next to the scroll bar.
– Adam Jaskiewicz Nov 24, 2008 at 17:53

9

Use Source control.

No matter how long it takes to set up (seconds..), it will always make your life easier! (still it's not OOP related).

Share Improve this answer

Follow

answered Nov 24, 2008 at 14:38

Tigraine

23.6k ● 12 ● 72 ● 113

I was thinking more of the code itself, but yes very important, and actually saves you time rather than costing time. – Iain Nov 24, 2008 at 14:42

On a related note, common coding styles and practices in relation to source control is important. I multiple developers write code in a similar style, conflict resolution is easier to handle, as well. – Joseph Ferris Nov 24, 2008 at 15:00

SC is the most important advice, I even use it at home for my personal project. Sometimes you will test something that will imply a lot of modifs. And then you love the revert. – Gilles Nov 24, 2008 at 15:05

9

Naming. Under pressure you'll write horrible code that you won't have time to document or even comment. Naming variables, methods and classes as explicitly as possible takes almost no additional time and will make the mess readable when you must fix it. From an OOP point of view, using nouns for classes and verbs for methods naturally helps encapsulation and modularity.

Share Improve this answer Follow

edited Nov 24, 2008 at 16:54
Dave Sherohman
46.1k ● 14 ● 66 ● 103

answered Nov 24, 2008 at 15:31
Yann Semet
623 ● 2 ● 5 ● 12

7

Unit tests - helps you sleep at night :-)

Share Improve this answer Follow

answered Nov 24, 2008 at 14:28
gef
7,124 ● 4 ● 44 ● 49

I don't think unit tests are appropriate to the kind of User Interface coding I do - am I wrong? – Iain Nov 24, 2008 at 14:47

Sorry didn't realised you meant UI. For integration test of the UI you may want to look at watin and selenium web applications. ASP.net mvc will also allow you to separate concerns and test.If you want to maintain your project with confidence then add tests at earliest opportunity! :) Good luck :-) – gef Nov 24, 2008 at 18:01

Also if you're using for Jquery - use QUnit docs.jquery.com/QUnit I agree with Joseph Ferris edited response below - maintenance time is usually the most amount of time you will ever spend on a project. The cost of change can be immense - tests reduce this pain and keep you sane! – gef Nov 24, 2008 at 18:38

5

This is rather obvious (I hope), but at the very least I always make sure my public interface is as correct as possible. The internals of a class can always be refactored later on.

Share Improve this answer Follow

answered Nov 24, 2008 at 14:34

user39603
2,235 ● 1 ● 16 ● 13

3

no public class with mutable public variables (struct-like).

Before you know it, you refer to this public variable all over your code, and the day you decide this field is a computed one and must have some logic in it... the refactoring gets messy.

If that day is before your release date, it gets messier.

Share Improve this answer

answered Nov 24, 2008 at 14:21

Follow

VonC
1.3m ● 558 ● 4.7k ● 5.6k

3

Think about the people (may even be your future self) who have to read and understand the code at some point.

Share Improve this answer

Follow

answered Nov 24, 2008 at 14:59

Brian Rasmussen
116k ● 34 ● 224 ● 323

2

[insert boilerplate not-OOP specific caveat here]

- Separation of concerns, unit tests, and that feeling that if something is too complex it's probably not conceptualised quite right yet.
- UML sketching: this has clarified and saved any amount of wasted effort so many times. Pictures are great aren't they? :)
- Really thinking about is-a's and has-a's. Getting this right first time is so important.

Share Improve this answer

Follow

answered Nov 24, 2008 at 17:12

annakata
75.7k ● 18 ● 115 ● 180

Unfortunately point 1 and 2 take way to long for me to be able to do. Point 3 - yes. – Iain Nov 24, 2008 at 17:25

2

Application of the single responsibility principal. Effectively applying this principal generates a lot of positive externalities.

Share Improve this answer Follow

answered Nov 24, 2008 at 17:31

Kevin McMahon
2,644 ● 25 ● 31

2

Like everyone else, not as much OOP practices, as much as practices for coding that apply to OOP.

1. Unit test, unit test, unit test. Defined unit tests have a habit of keeping people on task and not "wandering" aimlessly between objects.
2. Define and document all hierarchical information (namespaces, packages, folder structures, etc.) prior to writing production code. This helps to flesh out object relations and expose flaws in assumptions related to relationships of objects.
3. Define and document all applicable interfaces prior to writing production code. If done by a lead or an architect, this practice can additionally help keep more junior-level developers on task.

There are probably countless other "shoulds", but if I had to pick my top three, that would be the list.

***Edit in response to comment:* This is precisely why you need to do these things up front. All of these sorts of practices make continued maintenance easier. As you assume more risk in the kickoff of a project, the more likely it is that you will spend more and more time maintaining the code. Granted, there is a larger upfront cost, but building on a solid foundation pays for itself. Is your obstacle lack of time (i.e. having to maintain other applications) or a decision from higher up? I have had to fight both of those fronts to be able to adopt these kinds of practices, and it isn't a pleasant situation to be in.**

Share Improve this answer Follow

edited Nov 24, 2008 at 18:09

answered Nov 24, 2008 at 14:59

Joseph Ferris
12.7k ● 3 ● 48 ● 73

That's basically a list of things I don't have time to do. To do all that I'd need double the time. – Iain Nov 24, 2008 at 17:22

2

Of course everything should be Unit tested, well designed, commented, checked into source control and free of bugs. But life is not like that.

My personal ranking is this:

1. Use source control and actually write commit comments. This way you have a tiny bit of documentation should you ever wonder "what the heck did I think when I wrote this?"
2. Write clean code *or* document. Clean well-written code should need little documentation, as it's meaning can be grasped from reading it. Hacks are a lot different. Write why you did it, what you do and what you'd like to do if you had the time/knowledge/motivation/... to do it right
3. Unit Test. Yes it's down on number three. Not because it's unimportant but because it's useless if you don't have the other two at least halfway complete. Writing Unit tests is another level of documentation what you code should be doing (among others).
4. Refactor before you add something. This might sound like a typical "but we don't have time for it" point. But as with many of those points it usually saves more time than it costs. At least if you have at least some experience with it.

I'm aware that much of this has already been mentioned, but since it's a rather subjective matter, I wanted to add my ranking.

Share Improve this answer

Follow

answered Nov 24, 2008 at 18:39

Joachim Sauer

308k ● 59 ● 565 ● 620

1

No matter how fast a company wants it, I pretty much always try to write code to the best of my ability.

I don't find it takes any longer and usually saves a lot of time, even in the short-term.

I've can't remember ever writing code and never looking at it again, I always make a few passes over it to test and debug it, and even in those few passes practices like refactoring to keep my code DRY, documentation (to some degree), separation of concerns and cohesion all seem to save time.

This includes crating many more small classes than most people (One concern per class, please) and often extracting initialization data into external files (or arrays) and writing little parsers for that data... Sometimes even writing little GUIs instead of editing data by hand.

Coding itself is pretty quick and easy, debugging crap someone wrote when they were "Under pressure" is what takes all the time!

Share Improve this answer

Follow

answered Nov 24, 2008 at 17:16

Bill K

62.8k ● 18 ● 112 ● 158

1

At almost a year into my current project I finally set up an automated build that pushes any new commits to the test server, and man, I wish I had done that on day one. The biggest mistake I made early-on was going dark. With every feature, enhancement, bug-fix etc, I had a bad case of the "just one mores" before I would let anyone see the product, and it literally spiraled into a six month cycle. If every reasonable change had been automatically pushed out it would have been harder for me to hide, and I would have been more on-track with regard to the stakeholders' involvement.

Share Improve this answer Follow

answered Nov 24, 2008 at 18:23

Rex Miller
2,736 ● 1 ● 19 ● 26

1

Go back to code you wrote a few days/weeks ago and spend 20 minutes reviewing your own code. With the passage of time, you will be able to determine whether your "off-the-cuff" code is organized well enough for future maintenance efforts. While you're in there, look for refactoring and renaming opportunities.

I sometimes find that the name I chose for a function at the outset doesn't perfectly fit the function in its final form. With refactoring tools, you can easily change the name early before it goes into widespread use.

Share Improve this answer Follow

answered Nov 24, 2008 at 18:45

Larsenal

51.1k ● 43 ● 154 ● 223

1

Just like everybody else has suggested these recommendations aren't specific to OOP:

Ensure that you comment your code and use sensibly named variables. If you ever have to look back upon the quick and dirty code you've written, you should be able to understand it easily. A general rule that I follow is; if you deleted all of the code and only had the comments left, you should still be able to understand the program flow.

Hacks usually tend to be convoluted and un-intuitive, so some good commenting is essential.

I'd also recommend that if you usually have to work to tight deadlines, get yourself a code library built up based upon your most common tasks. This will allow you to "join the dots" rather than reinvent the wheel each time you have a project.

Regards,

Docta

Share Improve this answer Follow

edited Nov 24, 2008 at 18:47

answered Nov 24, 2008 at 15:36

Doctor Jones
21.7k ● 13 ● 80 ● 102

1

An actual OOP practice I always make time for is the [Single Responsibility Principle](), because it becomes so much harder to properly refactor the code later on when the project is "live".
By sticking to this principle I find that the code I write is easily re-used, replaced or rewritten if it fails to match the functional or non-functional requirements. When you end up with classes that have multiple responsibilities, some of them may fulfill the requirements, some may not, and the whole may be entirely unclear.

These kinds of classes are stressful to maintain because you are never sure what your "fix" will break.

Share Improve this answer Follow

answered Nov 25, 2008 at 10:50
Joris Timmermans
11k ● 2 ● 52 ● 78

0

For this special case (short deadlines and with lots of code that will never be used again) I suggest you to pay attention to embedding some script engine into your OOP code.

Share Improve this answer Follow

answered Nov 24, 2008 at 14:53
macropas
3,150 ● 3 ● 23 ● 25

Is this to cut down the amount of compilation? – Iain Nov 24, 2008 at 17:23

Not only compilation time minimization and source code shrinking, but elelements of DSL (for example, C++ and TCL or Java and JScheme). – macropas Nov 24, 2008 at 18:40

0

Learn to "refactor as-you-go". Mainly from an "extract method" standpoint. When you start to write a block of sequential code, take a few seconds to decide if this block could stand-alone as a reusable method and, if so, make that method immediately. I recommend it even for throw-away projects (especially if you can go back later and compile such methods into your personal toolbox API). It doesn't take long before you do it almost without thinking.

Hopefully you do this already and I'm preaching to the choir.

Share Improve this answer Follow

answered Nov 24, 2008 at 18:22

Lewis Baumstark
121 ● 5