

What does it take to be a better OO programmer? [closed]

Asked 16 years, 2 months ago Modified 8 years, 4 months ago


Viewed 2k times



11



Closed. This question is [opinion-based](#). It is not currently accepting answers.

 **Want to improve this question?** Update the question so it can be answered with facts and citations by [editing this post](#).

Closed 11 years ago.

[Improve this question](#)

I've almost 6 years of experience in application development using .net technologies. Over the years I have improved as a better OO programmer but when I see code written by other guys (especially the likes of Jeffrey Richter, Peter Golde, Ayende Rahien, Jeremy Miller etc), I feel there is a generation gap between mine and their designs. I usually design my classes on the fly with some help from tools like ReSharper for refactoring and code organization.

So, my question is "what does it takes to be a better OO programmer". Is it

- a) Experience
- b) Books (reference please)
- c) Process (tdd or uml)
- d) patterns
- e) anything else?

And how should one validate that the design is good, easy to understand and maintainable. As there are so many buzzwords in industry like dependency injection, IoC, MVC, MVP, etc where should one concentrate more in design. I feel abstraction is the key. What else?

.net

oop

Share

Improve this question

Follow

edited Oct 24, 2008 at 9:42



Mendelt

37.5k ● 6 ● 75 ● 97

asked Oct 24, 2008 at 6:57




Sunil

169 ● 1 ● 8

-
- 9 Paraphrasing Paul Graham, most developers look at the rest of the industry and see three groups: Newbies, Peers and "oddballs doing weird stuff." Look around for the oddballs and work out what they're doing, and why. – Bevan Oct 24, 2008 at 9:09
-

f) practice g) introspection h) review i) reading code j) experimentation i) modeling j) iteration k) refactoring ...
– [Steven A. Lowe](#) Sep 16, 2010 at 18:36

For book references, check out [What is the single most influential book every programmer should read?](#) and choose your pick. – [madkris24](#) Dec 19, 2010 at 18:29 

19 Answers

Sorted by:

Highest score (default)



33

you'll probably find that the elegant OO designs that you admire *are not the first iteration*, but result from several adjustments, refactorings, and fine-tunings



try to qualify why you think their designs are 'better' than yours, and adjust accordingly



the difference between an amateur writer and a professional writer is that the professional *rewrites*; the same holds for programming

Share Improve this answer

answered Oct 24, 2008 at 7:00

Follow



[Steven A. Lowe](#)

61.1k ● 19 ● 135 ● 204

2 I believe! ;) No, really. My wife is taking a sculpture class and her professor keeps telling everyone that the best thing that can happen to their sculptures is to have them fall of the clay stand. Reason being is that the next time they redo it, it will be soo much better. Thanks for the great input. – [Mike Grace](#) Sep 23, 2009 at 4:47

- 1 Totally agree with the last line. This is why I am an advocate of "throw away prototyping" Make it a rule that the first iteration gets tossed. – [Neil N](#) Sep 1, 2010 at 15:12
-



[humor]

11



Object oriented skills can be learned from books and other resources. But if you are lucky, you inherit the skills from your parent. Most of the time it is a matter to provide and use the correct method. Be careful about the amount of arguments. Less is better.



Use the right names for anything. Use verbs as a method of activity. Use nouns for anything that needs to be remembered. Don't be too creative and keep your solution as simple as possible, else your users will be confusers.

It is also important to encapsulate the nasty details. And be sure to hide your private members for the general public else the unexpected behaviour will occur. Be also aware to catch your exceptional situation at the right level.

Rest me to press you to always test your units and to use the right interface to provide just enough handles for the happy user.

[/humor]

Follow

answered Oct 24, 2008 at 7:29



Toon Krijthe

53.4k ● 38 ● 149 ● 202

11 I often fail to encapsulate private members. Perhaps that's why I get these nasty stares at the subway? – [gnud](#) Oct 24, 2008 at 9:27



7

A little bit of everything. As for any language (verbal ou programming), the more you'll get exposed to it, the more you'll learn.



So read books, read your coworkers code. And at least as much important, learn new programming languages: they will broaden your vision, make you more critical of your own code and allow you to rethink your programming habits.



About design patterns, they are a de-facto standard way to work around common problems in common languages. You must know them to avoid reinventing the wheel and better communicate with your coworkers, but you should also see them as working around missing features in the languages you are using. The state machine pattern exists only in languages that don't provide them as builtins (not that I know a language that provides them, but you get the picture).

I would also add:

- always refactor if needed and time permits (harmless since you have unit tests to avoid regressions, of course).
- learn when to avoid inheritance (which is more often than you think).
- learn when to avoid OO (when it doesn't add any value).
- don't confuse OO with encapsulation (which is the main benefit of OO but is also provided by other paradigms).

Share Improve this answer

edited Oct 24, 2008 at 8:08

Follow

answered Oct 24, 2008 at 7:45



bltxd

9,103 ● 5 ● 33 ● 44



5



Good OO design:

- it reads like poetry
- doesn't need any comment
- trust your objects (let the control go)
- favor composition over inheritance



Share Improve this answer

edited Aug 21, 2016 at 15:05

Follow

answered Oct 24, 2008 at 9:02



eddy147

4,973 ● 8 ● 41 ● 61

what does the last bullet mean? – Zhang18 Sep 1, 2010 at 15:17

- 2 One must always strive for loosely coupled objects. The strongest coupling is a parent and child. This makes it difficult to re-use code. So instead of using inheritance between concrete objects use composition instead. An excellent article about this is here: berniecode.com/writing/inheritance – eddy147 Sep 3, 2010 at 7:40
-



4

You become a better object oriented programmer by forgetting object orientation for moment and orient yourself to writing cleaner, better programs while improving your existing programs.



Share Improve this answer

answered Oct 24, 2008 at 7:04



Follow



Cheery

25.4k ● 16 ● 61 ● 83



4

First books, you'll need to know some of the GoF patters but more importantly you need to understand the principles behind the patterns. Understand the differences between old-style (use inheritance for code-reuse) vs. new style (prefer encapsulation over inheritance) oo design. Two good books to read are Design Patterns Explained by Shalloway and Trott and Agile Software





Development, Principles, Patterns, and Practices by Bob Martin.

Then you need experience. The theory in books is nice but you need to fine tune your sense of when to use what. How to use process to fine tune your designs (Steven A. Lowe already named iterations) lots of old-timey-oo-gurus described iterative programming and oo-programming in the same papers and books.

And last but I think most importantly you need feedback and communication. Talk to other programmers preferably outside the organization you work in. Try to work with as many people as possible (OSS is nice for that) eventually you learn from people not books.

Share Improve this answer

edited Oct 24, 2008 at 8:42

Follow

answered Oct 24, 2008 at 7:09



Mendelt

37.5k ● 6 ● 75 ● 97

1 inheritance vs encapsulation is not a new-style vs old-style argument, this argument is probably as old as OOP [Simula, 1966]! – [Steven A. Lowe](#) Oct 24, 2008 at 7:43

I agree these aren't new ideas. The reason I call them new-style is because they have only started to become common-practice in the last 10 years or so. If you look at older oo-books (especially those targeted at C++ developers) you'll see a lot of inheritance-for-reuse type of design. – [Mendelt](#) Oct 24, 2008 at 8:38



3



Practice functional programming, in both dedicated functional programming languages, and object-oriented languages. This will increase your appreciation of how reusable algorithms help encourage well-defined interfaces, which leads to easier-to-work-with programme elements.



Share Improve this answer

answered Oct 24, 2008 at 7:30

Follow



[Marcin](#)

49.8k ● 18 ● 132 ● 206



3



Besides learning from academic stuff like books and papers, I **highly** recommend: learn more than one language, specially if you come from a Java/C# mainstream. Learn ruby, learn groovy, learn smalltalk, learn lisp, learn the differences between them both in theory and in practice.



An academic but excellent example is the single vs. multiple dispatch: you can check the [wikipedia entry](#) and see for yourself how you would write different code depending on language capabilities. More fundamentally, this helps you understand how to achieve the same effects in language X while maintaining a solid design.

The key here is experimenting, understanding and evolving. You also learn a lot from reading or helping out in some open source projects, they typically have good architecture and implementation (at least the big ones).

Share Improve this answer

answered Oct 24, 2008 at 9:37

Follow



Miguel Ping

18.3k ● 23 ● 91 ● 137



1



Hello and good day for everybody

Like Cheery said: "you become a better object oriented programmer by forgetting object orientation for moment and orient yourself to writing cleaner, better programs while improving your existing programs".

Thats the key : Think and get the simplest solution as possible and code similar

Thats all With no more.... bye bye

Share Improve this answer

answered Oct 24, 2008 at 8:50

Follow



yeradis

5,347 ● 5 ● 27 ● 27



1



Something that's worked for me is **Reading**. I just had a Bulb moment with this book... [David West's Object Thinking](#) which elaborates [Alan Kay's comment of 'The object revolution has yet to happen'](#). OO is different things to different people.. couple that with the fact that your tools influence how you go about solving a problem. So learn multiple languages.

[Object Thinking David West http://ecx.images-amazon.com/images/I/51hmvXjQtL._SL500_BO2,204,203,200_AA219_PIsitb-sticker-dp-arrow,TopRight,-24,-23_SH20_OU01_.jpg](http://ecx.images-amazon.com/images/I/51hmvXjQtL._SL500_BO2,204,203,200_AA219_PIsitb-sticker-dp-arrow,TopRight,-24,-23_SH20_OU01_.jpg)

Personally I think understanding philosophy, principles and values behind a practice rather than mimic-ing a

practice helps a lot.

Share Improve this answer

answered Oct 24, 2008 at 9:38

Follow



Gishu

137k ● 47 ● 226 ● 311



1



What works for me:

- Know the domain of your app, and stay close to the domain as long as possible while still avoiding code duplication.
- Don't stick to one language, learn several OO language. Smalltalk, Comon Lisp, Python, Javascript all have very interesting ways of implementing OOP. Browse their source (the Smalltalk Object Browser is a great tool for that).
- Implement an OOP lib in a language that has no OOP standard, like Lua: this will show you that `self/this` can be nothing more than an implicit first argument pointing to the state, and delegating its behavior to its `class/vtable/metatable` (which can in turn delegate the call to its parent `class`).

Cheers!

Share Improve this answer

answered Oct 24, 2008 at 9:56

Follow



Sébastien RoccaSerra

17.2k ● 8 ● 51 ● 54



1



To have your design reviewed by someone is quite important. To review and maintain legacy code helps you to realize what makes the software rotten. Thinking is also very important; One one hand don't rush into implementing the first idea. On the other hand, don't think everything at once. Do it iteratively.

Regular reading of books/articles, like Eric Evan's [Model Driven Design](#), or learning new languages (Smalltalk, Self, Scala) that take different approach to OO, helps you to really understand.

Software, and OO, is all about abstractions, responsibilities, dependencies and duplication (or lack of it). Keep them on your mind on your journey, and your learning will be steady.

Share Improve this answer

Follow

answered May 17, 2009 at 7:35



egaga

21.7k ● 10 ● 47 ● 61



1



I think that for a good OO design one needs to acquire the proper mindset.

I am not sure what the rationale was behind the person downgrading the answers referring to Smalltalk, but Smalltalk sure has something to do with it. The keyword is metaphors. For most programmers, and this is actually enforced in most computer science education, the central metaphors is mechanical: clockworks or steam engines.

The hardest effort is that you need to replace this metaphor with another. Not a mechanical metaphor, but the metaphor of living systems, of ecologies. If you have put effort in acquiring some kind of excellence in the first metaphor, this can be really difficult and even painful.

The way in which all those objects in Smalltalk work together closely follows that metaphor. I also found the book by Kevin Kelly: Out of Control interesting in this context, because the book contains many examples of real solutions based on this metaphor.

Share Improve this answer
Follow

answered Sep 1, 2010 at 15:11



[robject](#)

808 ● 1 ● 6 ● 12



0

knowing and understanding the GoF (Gamma et al.) design patterns is never wrong. they can be applied to a lot of situations. i guess they are essential stuff for every developer!



Share Improve this answer
Follow

answered Oct 24, 2008 at 7:01



[Joachim Kerschbaumer](#)

9,841 ● 7 ● 51 ● 84



0

TDD worked for me. Writing the tests first without good oo is a real PIA so I had to become better at it.

Share Improve this answer

answered Oct 24, 2008 at 13:42



Follow



Dala

1,889 ● 2 ● 20 ● 22





0

Being well mentored, having those people sit down with you when you have a problem and work it through with you. You gain an insight into how they are thinking of things in an OO way.



Regular code reviews. Think I learnt most from a team code review conducted via PowerPoint in which some scorn was poured on my code. Concentrated my mind, it did.

Helped of course that I learned OO with Smalltalk (everyone should be **forced** to do this IMHO). Language always encouraged the concept of 'message sending'. That is, don't inline the stuff here, write a new method and call it (ie send a message and get an answer**) result? Simplicity, low coupling, high cohesion.

** When I write my JavaDoc for an accessor method I still write 'answer a [blah] that represents the [thing]' People may think I'm strange, but no-ones called the cops so far....

[Share](#) [Improve this answer](#)

[Follow](#)

answered Nov 5, 2008 at 12:31



[Chris Brooks](#)

299 ● 1 ● 2



0

It takes being a better programmer to be a better OO programmer.



OO has been evolving over the years, and it has a lot to do with changing paradigms and technologies like n-tier architecture, garbage collection, Web Services, etc.. the kind of things you've already seen. There are fundamental principles such as maintainability, reusability, low coupling, KISS, DRY, Amdahl's law, etc. you have to learn, read, experience, and apply it yourself.

OO is not an end on its own, but rather a means to achieve programming solutions. Like games, sports, and arts, practices cannot be understood without principles; and principles cannot be understood without practices.

To be more specific, here are some of the skills that may make one a better programmer. Listen to the domain experts. Know how to write tests. Know how to design a GUI desktop software. Know how to persist data into database. Separate UI layer and logic layer. Know how to write a class that acts like a built-in class. Know how to write a graphical component that acts like a built-in component. Know how to design a client/server software. Know networking, security, concurrency, and reliability.

[Design patterns](#), MVC, [UML](#), [Refactoring](#), [TDD](#), etc. address many of the issues, often extending OO in creative ways. For example, to decouple UI layer dependencies from logic layer, an interface may be introduced to wrap the UI class. From pure object-oriented point of view, it may not make much sense, but it makes sense from the point of view of separation of UI layer and logic layer.

Finally, realizing the limitations of OO is important too. In modern application architecture, the purist data + logic view of OO doesn't always mesh very well. Data transfer object ([Java](#), [MS](#), [Fowler](#)) for example intentionally strips away logic part of the object to make it carry only the data. This way the object can turn itself into a binary data stream or XML/JSON. The logic part may be handled both at client and server side in some way.

Share Improve this answer

answered May 17, 2009 at 8:33

Follow



[Eugene Yokota](#)

95.5k ● 45 ● 217 ● 320



0



It suddenly started to make sense to me when I tried to reduce dependencies between classes, (java) packages and modules, to specify clear module boundaries and to remove cycles in dependencies. Especially this cycles removal brought many insights to me. Your class/packages/module dependencies should ideally form a tree.

Share Improve this answer

edited May 17, 2009 at 8:34

Follow

answered May 17, 2009 at 8:11



[Peter Štibraný](#)

32.9k ● 16 ● 92 ● 116



3 is the magic number; no less no more and it is everywhere.

0



Less possible inheritance by minimizing sub-classing.
Much possible loose coupling by having independent objects.



Stick on it.



Explaining by sample.

First step interface is the access definition to your system.

```
interface i { getObject($o); }
```

- In terms of Web page is this the unique URL or the parameter.
- In terms of MVC is this the Model.
- In terms of Tree is this the connection (branch) from parent to child.

Second step abstraction is the control to your system.

```
abstract class Controller {  
    function getObject(){}  
    function validateObject(){  
  
        // o must be a digit  
        // o must be between 0 and 120  
        // ... etc  
    }  
}
```

- In terms of Web page is this the Validation of the requirements like User logged in, Database connected etc.
- In terms of MVC is this the Controller.
- In terms of Tree is this the parent.

Third and last step is real object that should knit and cultivate the available data.

```
class View extends Controller (InputObject $in, OutputObject $out) {
```

The available data in this concrete class comes also from 3 directions.

- extends Controller
- InputObject \$in
- OutputObject \$out

This presentation object would be - In terms of Web page the output of the result, the page content. - In terms of MVC the view. - In terms of Tree the leaf.

I recommend you to think off more of this Triangle sets. Try for instance

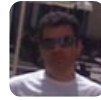
- the setup of the Planets and their orbit behavior within our solar system.
- structure of a Car and its behavior when it will ride.

- reason and result within different statuses.

Share Improve this answer

answered May 17, 2012 at 14:06

Follow



Webist

37 ● 1 ● 9
