

# Aspect Oriented Programming vs. Object-Oriented Programming

Asked 16 years, 2 months ago   Modified 7 months ago

Viewed 66k times



234



Like most developers here and in the entire world, I have been developing software systems using object-oriented programming (OOP) techniques for many years. So when I read that aspect-oriented programming (AOP) addresses many of the problems that traditional OOP doesn't solve completely or directly, I pause and think, is it real?

I have read a lot of information trying to learn the keys of this AOP paradigm and I'm in the same place, so, I wanted to better understand its benefits in real world application development.

Does somebody have the answer?

oop

aop

paradigms

Share

Improve this question

Follow

edited Sep 19, 2011 at 21:29



Erik B

42.5k ● 27 ● 125 ● 143

asked Oct 24, 2008 at 9:00



yeradis

5,347 ● 5 ● 27 ● 27

- 16 All the answers have been very good, this is a perfect case for a single community edited answer that would combine them all. All are saying the same thing but in different ways and using different examples that add to the overall value – [Vinko Vrsalovic](#) Oct 24, 2008 at 9:29

8 Answers

Sorted by:

Highest score (default)



378



Why "vs"? It is not "vs". You can use Aspect Oriented programming in combination with functional programming, but also in combination with Object Oriented one. It is not "vs", it is "Aspect Oriented Programming **with** Object Oriented Programming".



To me, AOP is some kind of "meta-programming". Everything that AOP does could also be done without it by just adding more code. AOP just saves you from writing this code.

Wikipedia has one of the best examples for this meta-programming. Assume you have a graphical class with many "set...()" methods. After each set method, the data of the graphics changed, thus the graphics changed and thus the graphics need to be updated on screen. Assume to repaint the graphics you must call "Display.update()". The classical approach is to solve this by adding *more code*. At the end of each set method, you write

```
void set...(...) {  
    :  
    :  
    Display.update();  
}
```

If you have 3 set-methods, that is not a problem. If you have 200 (hypothetical), it's getting real painful to add this everywhere. Also, whenever you add a new set-method, you must be sure to not forget adding this to the end, otherwise you just created a bug.

AOP solves this without adding tons of code, instead you add an aspect:

```
after() : set() {  
    Display.update();  
}
```

And that's it! Instead of writing the update code yourself, you just tell the system that after a set() pointcut has been reached, it must run this code and it will run this code. No need to update 200 methods, no need to make sure you don't forget to add this code on a new set-method. Additionally, you just need a pointcut:


```
pointcut set() : execution(* set*(*) ) &&  
    this(MyGraphicsClass) && within(com.company.*);
```

What does that mean? That means if a method is named "set\*" (\* means any name might follow after set), regardless of what the method returns (first asterisk) or

what parameters it takes (third asterisk) *and* it is a method of MyGraphicsClass *and* this class is part of the package "com.company.\*", then this is a set() pointcut. And our first code says "*after* running any method that is a set pointcut, run the following code".

See how AOP elegantly solves the problem here? Actually, everything described here can be done at compile time. An AOP preprocessor can just modify your source (e.g. adding Display.update() to the end of every set-pointcut method) before even compiling the class itself.

However, this example also shows one of the big downsides of AOP. AOP is actually doing something that many programmers consider an "[Anti-Pattern](#)". The exact pattern is called "[Action at a distance](#)".



Action at a distance is an anti-pattern (a recognized common error) in which behavior in one part of a program varies wildly based on difficult or impossible to identify operations in another part of the program.

As a newbie to a project, I might just read the code of any set-method and consider it broken, as it seems to not update the display. I don't **see** by just looking at the code of a set-method, that after it is executed, some other code will "magically" be executed to update the display. I consider this a serious downside! By making changes to a method, strange bugs might be introduced. Further

understanding the code flow of code where certain things seem to work correctly, but are not obvious (as I said, they just magically work... somehow), is really hard.

## Update

Just to clarify that: Some people might have the impression I'm saying AOP is something bad and should not be used. That's not what I'm saying! AOP is actually a great feature. I just say "Use it carefully". AOP will only cause problems if you mix up normal code and AOP for the same *Aspect*. In the example above, we have the Aspect of updating the values of a graphical object and painting the updated object. That is in fact a single aspect. Coding half of it as normal code and the other half of it as aspect is what adds the problem.

If you use AOP for a completely different aspect, e.g. for logging, you will not run into the anti-pattern problem. In that case a newbie to the project might wonder "Where do all these log messages come from? I don't see any log output in the code", but that is not a huge problem. Changes he makes to the program logic will hardly break the log facility and changes made to the log facility will hardly break his program logic - these aspects are totally separated. Using AOP for logging has the advantage that your program code can fully concentrate on doing whatever it should do and you still can have sophisticated logging, without having your code being cluttered up by hundreds of log messages everywhere. Also when new

code is introduced, magically log messages will appear at the right time with the right content. The newbie programmer might not understand why they are there or where they came from, but since they will log the "right thing" at the "right time", he can just happily accept the fact that they are there and move on to something else.

So a good usage of AOP in my example would be to always log if any value has been updated via a set method. This will not create an anti-pattern and hardly ever be the cause of any problem.

One might say, if you can easily abuse AOP to create so many problems, it's a bad idea to use it all. However, which technology can't be abused? You can abuse data encapsulation, you can abuse inheritance. Pretty much every useful programming technology can be abused. Consider a programming language so limited that it only contains features that can't be abused; a language where features can only be used as they were initially intended to be used. Such a language would be so limited that it's arguable if it can be even used for real world programming.

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

[Follow](#)

[edited May 13 at 2:05](#)



[Pang](#)


**10.1k** ● 146 ● 85 ● 124

[answered Oct 24, 2008 at 9:21](#)



[Mecki](#)

**133k** ● 34 ● 259 ● 269

- 
- 7 Logging seems to be one specific example where AOP doesn't result in Action at a Distance. At this time, wikipedia as the example of using aspect for things like security checks, which really do make the program flow that much more to understand. – [kizzx2](#) Sep 25, 2010 at 7:22 
- 
- 9 @kizzx2: Great point on logging, in fact -- it's the best example I've seen so far AOP's strength, without knowing to much about AOP. Thanks for sharing! – [blunders](#) Nov 30, 2010 at 1:25
- 
- 5 @Pacerier My example is simplified because SO is no a teaching forum. I was just answering the question of the questioner, probably far more detailed than would have been necessary. If you want to know more about AOP, try reading some programmer documentation and if you have a detailed question, why not asking it here? No, not in a comment, go and create a new question because *that* is what SO is all about. I'm sure someone will be able to dispel your doubts in a reply. – [Mecki](#) Jun 20, 2014 at 16:20
- 
- 3 @Pacerier Sorry, but I fail to see your point. See here: [stackoverflow.com/a/8843713/15809](http://stackoverflow.com/a/8843713/15809) This code logs every call to every public method including all method argument types and values. You write this exactly once and there is zero boilerplate code added to any method, it's just the code shown in the answer. – [Mecki](#) Jul 1, 2014 at 10:10
- 
- 1 Great explanation, i agree that not being able to see what is executed right away in the code is a very big downside. It's not like crystal clear. I prefer to see `Display.update()` rather than thinking something like: "the display is being updated but where?". If i know it is using AOP, i might also need to do the extra-job of searching where the code i'm seeing is being advised, isn't that right? I believe IDE's help, but i still think seeing `Display.update()` is better. There is a pattern called Template method that could solve that kind of problem i



39



Aspect oriented programming provides a nice way to implement cross-cutting concerns like logging, security.

These cross-cutting concerns are pieces of logic that have to be applied at many places but actually don't have anything to do with the business logic.

You shouldn't see AOP as a replacement of OOP, but more as a nice add-on that makes your code more clean, loosely-coupled and focused on the business logic.

So by applying AOP you will get 2 major benefits:

1. The logic for each concern is now in one place, as opposed to being scattered all over the code base.
2. Classes are cleaner since they only contain code for their primary concern (or core functionality) and secondary concerns have been moved to aspects.

Share Improve this answer

Follow

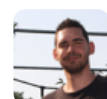
edited Nov 19, 2020 at 10:21



X X

81 ● 7

answered Oct 24, 2008 at 9:11



[nkr1pt](#)

4,666 ● 5 ● 38 ● 55





33



OOP and AOP are not mutually exclusive. AOP can be good addition to OOP. AOP is especially handy for adding standard code like logging, performance tracking, etc. to methods without clogging up the method code with this standard code.



Share Improve this answer

answered Oct 24, 2008 at 9:09



Follow



Norbert B.

5,730 ● 3 ● 27 ● 30



14



I think there is no general answer to this question but one thing to be noted is, that AOP does not *replace* OOP but adds certain decomposition features that address the so-called *tyranny of the dominant composition* ([1](#)) (or crosscutting concerns).



It surely helps in certain cases as long as you're in control of the tools and languages to use for a specific project, but also adds a new level of complexity regarding interaction of aspects and the need for additional tools like the [AJDT](#) to still understand your program.



Gregor Kiczales once gave an interesting introductory talk on AOP at Google Tech Talks which I recommend watching: [Aspect Oriented Programming: Radical Research in Modularity](#).

Share Improve this answer

edited Sep 29, 2014 at 7:39

Follow

answered Oct 24, 2008 at 9:16



fhe

6,197 ● 1 ● 43 ● 45



9



First of all AOP will not replace OOP. AOP extends OOP. The ideas and practices of OOP stay relevant. Having a good object design will probably make it easier to extend it with aspects.

I think the ideas that AOP brings are important. We need to work out ways to implement cross-cutting-concerns over different classes in your program without having to change the classes themselves. But I think the AOP will eventually just become part of other tools we use and not a separate tool or technique. We already see this happening.

A couple of dynamic languages like Ruby and Python have language constructs like mixins that solve the same problems. This looks a lot like AOP but is better integrated in the language.

Spring and Castle and a couple of other dependency injection framework have options to add behaviour to the classes they inject. This is a way of doing runtime-weaving and I think this has a lot of potential.

I don't think you'll have to learn a completely new paradigm to use AOP. The ideas are interesting but are slowly being absorbed by existing tools and languages. Just stay informed and try out these tools.

Share Improve this answer

answered Oct 24, 2008 at 9:16

Follow



Mendelt

37.5k ● 6 ● 75 ● 97



9



**OOP** is mainly used to organise your **business logic** while **AOP** helps to organise your **non-functional things** like Auditing, Logging, Transaction Management, Security etc.

This way you can decouple your business logic from non-functional logic, which makes code cleaner.

Other advantage is you can apply the advice (such as auditing) very consistently, without implementing any interface, which gives great flexibility for modification without touching the business logic.

This way its easy to implement **Separation Of Concern** and **Single Responsibility**.

Moreover, its easy to port business logic from one framework (say Spring) to somewhere else (in future) when its only solve a business problem

Share Improve this answer

edited Jul 6, 2021 at 13:55

Follow



GPopat

455 ● 4 ● 16



0

AOP is a new programming paradigm dealing with this concept. An aspect is a software entity implementing a specific non-functional part of the application.



I think this article is a good place to start with Aspect Oriented Programming:



<http://www.jaftalks.com/wp/index.php/introduction-to-aspect-oriented-programming/>



Share Improve this answer

edited Jun 25, 2014 at 6:41

Follow



**Qing**

3,079 ● 7 ● 26 ● 31

answered Oct 13, 2010 at 11:30



**JafTalks**

33 ● 2

---

1 here lies a dead link – [john k](#) Sep 24 at 14:51

---



0

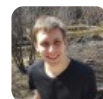
In one sentence: AOP is a paradigm that aims to separate business logic code from non-business logic code and decrease amount of boilerplate code, facilitating modularization and improved code maintenance.



Share Improve this answer

answered Jan 19 at 5:52

Follow



**Dmitrii Zyrianov**

2,318 ● 1 ● 23 ● 30