What's the point of OOP?

Asked 16 years, 4 months ago Modified 8 years, 5 months ago

Viewed 32k times

126

votes

As far as I can tell, in spite of the countless millions or billions spent on OOP education, languages, and tools, OOP has not improved developer productivity or software reliability, nor has it reduced development costs. Few people use OOP in any rigorous sense (few people adhere to or understand principles such as LSP); there seems to be little uniformity or consistency to the approaches that people take to modelling problem domains. All too often, the class is used simply for its syntactic sugar; it puts the functions for a record type into their own little namespace.

I've written a large amount of code for a wide variety of applications. Although there have been places where true substitutable subtyping played a valuable role in the application, these have been pretty exceptional. In general, though much lip service is given to talk of "re-use" the reality is that unless a piece of code does *exactly* what you want it to do, there's very little cost-effective "re-use".

It's extremely hard to design classes to be extensible *in the right way*, and so the cost of extension is normally so great that "re-use" simply isn't worthwhile.

In many regards, this doesn't surprise me. The real world isn't "OO", and the idea implicit in OO--that we can model things with some class taxonomy--seems to me very fundamentally flawed (I can sit on a table, a tree stump, a car bonnet, someone's lap--but not one of those is-a chair). Even if we move to more abstract domains, OO modelling is often difficult, counterintuitive, and ultimately unhelpful (consider the classic examples of circles/ellipses or squares/rectangles).

So what am I missing here? Where's the value of OOP, and why has all the time and money failed to make software any better?

language-agnostic **oop**

Share

edited Aug 23, 2008 at 16:00

community wiki
3 revs, 3 users 100%
DrPizza

11 Your analogy is too high an abstraction for your intended "use case"; table, tree stump, bonnet, someone's lap are compositions of molecules, atoms, protons, neutrons,

electrons, forming a large-enough surface area for your butt to rest against by force of gravity. – icelava Dec 31, 2008 at 2:47

38 No matter how many times this same thread is started, it always garners a lot of interest (despite the fact that duplicates are usually not tolerated here). And of course, the chosen answer is always one that agrees with the initial opinion of the asker. – TM. Dec 31, 2008 at 3:04

4 The problem with OOP is failure to put it in context. It is excellent for some purposes, not all purposes. It is a great tool. It is a lousy gospel. – Mike Dunlavey Dec 31, 2008 at 15:19

16 I'm sorry but I can't shake the feeling that you've never programmed using any kind of language. Here's why: OOP is the base of operation for base component libraries in all the modern environments (Java, .NET, Python, Ruby - just to name a few main-stream ones). All those base libraries are reused on a daily basis so if that doesn't count I don't know what does. So don't get me wrong here but code reuse if a fact - and an extremly common one! I don't want for this to sound offending in any way - just making a point here. – Matthias Hryniszak Aug 8, 2009 at 19:08

2 @George Jempty: It was Joel Spolsky in "How Microsoft lost the API war" (joelonsoftware.com/articles/APIWar.html), headline of the passage is "Automatic Transmissions Win the Day". – GodsBoss Feb 19, 2010 at 16:42

Comments disabled on deleted / locked posts / reviews |

45 Answers

Sorted by: Highest score (default)

1 2 Next

119

votes

> The real world isn't "OO", and the idea implicit in OO--that we can model things with some class taxonomy--seems to me very fundamentally flawed

While this is true and has been observed by other people (take Stepanov, inventor of the STL), the rest is nonsense. OOP may be flawed and it certainly is no silver bullet but it makes large-scale applications much simpler because it's a great way to reduce dependencies. Of course, this is only true for “good” OOP design. Sloppy design won't give any advantage. But good, decoupled design can be modelled very well using OOP and not well using other techniques.

There are much better, more universal models (Haskell's type model comes to mind) but these are also often more complicated and/or difficult to implement efficiently. OOP is a good trade-off between extremes.

Share

edited May 23, 2017 at 11:46

community wiki
3 revs
Konrad Rudolph

10 I find it interesting that this has more up votes than the question, and approved answer, combined. – Brad Gilbert Sep 15, 2008 at 16:14

15 I find Haskell's type system much more intuitive than OO. – axblount Oct 23, 2008 at 16:03

4 @Konrad: "but it makes large-scale applications much simpler" - hmmm, Im not sure that's true. It makes them more maintainable in the sense that chnaging one thing should not break another, but simpler? That's a hrad one to swallow... – Mitch Wheat Dec 31, 2008 at 1:14

2 @BradGilbert: of course the asker selected an answer which aligns perfectly with the opinion in his initial question. Which raises the question, why bother asking the question if you already have decided on your answer? – TM. Dec 31, 2008 at 4:55

2 @Mitch: I'd go as far as claiming that you *cannot* (as of today) write large-scale applications without some kind of object orientation. Large-scale here is > 1M LOC. – Konrad Rudolph Dec 31, 2008 at 13:30

45 votes

OOP isn't about creating re-usable classes, its about creating Usable classes.

Share

answered Aug 23, 2008 at 18:04

community wiki
Brian Leahy

1 Fair enough. But then it doesn't solve the "reinventing the wheel" problem, which is quite honestly a serious one in software development - instead of having one library which has N pairs of eyes looking for flaws we have N libraries with a pair of eyes doing so. There are so many usable classes you can

build before you need to start reusing them. And OOP in my educated opinion, is fundamentally flawed at exactly that area - code reuse. It hides data and "marries" it to methods, effectively making said data inaccessible or hardly interoperable. – Armen Michaeli Jun 9, 2013 at 10:35

42

votes

> All too often, the class is used simply for its syntactic sugar; it puts the functions for a record type into their own little namespace.

Yes, I find this to be too prevalent as well. This is not Object Oriented Programming. It's Object Based Programming and data centric programing. In my 10 years of working with OO Languages, I see people mostly doing Object Based Programming. OBP breaks down very quickly IMHO since you are essentially getting the worst of both words: 1) Procedural programming without adhering to proven structured programming methodology and 2) OOP without adhering to to proven OOP methodology.

OOP done right is a beautiful thing. It makes very difficult problems easy to solve, and to the uninitiated (not trying to sound pompous there), it can almost seem like magic. That being said, OOP is just one tool in the toolbox of programming methodologies. It is not the be all end all methodology. It just happens to suit large business applications well.

Most developers who work in OOP languages are utilizing examples of OOP done right in the frameworks and types

that they use day-to-day, but they just aren't aware of it. Here are some very simple examples: ADO.NET, Hibernate/NHibernate, Logging Frameworks, various language collection types, the ASP.NET stack, The JSP stack etc... These are all things that heavily rely on OOP in their codebases.

Share

answered Aug 23, 2008 at 16:42

community wiki
Daniel Auger

32

votes

Reuse shouldn't be a goal of OOP - or any other paradigm for that matter.

Reuse is a side-effect of an good design and proper level of abstraction. Code achieves reuse by doing something useful, but not doing so much as to make it inflexible. It does not matter whether the code is OO or not - we reuse what works and is not trivial to do ourselves. That's pragmatism.

The thought of OO as a new way to get to reuse through inheritance is fundamentally flawed. As you note the LSP violations abound. Instead, OO is properly thought of as a method of managing the complexity of a problem domain. The goal is maintainability of a system over time. The primary tool for achieving this is the separation of public interface from a private implementation. This allows us to

have rules like "This should only be modified using ..." enforced by the compiler, rather than code review.

Using this, I'm sure you will agree, allows us to create and maintain hugely complex systems. There is lots of value in that, and it is not easy to do in other paradigms.

Share

answered Sep 13, 2008 at 0:14

community wiki
theschmitzer

3 true about reuse and OO being separate concerns.
– Dan Rosenstark Dec 31, 2008 at 1:39

28
votes

Verging on religious but I would say that you're painting an overly grim picture of the state of modern OOP. I would argue that it actually **has** reduced costs, made large software projects manageable, and so forth. That doesn't mean it's solved the fundamental problem of software messiness, and it doesn't mean the average developer is an OOP expert. But the modularization of function into object-components has certainly reduced the amount of spaghetti code out there in the world.

I can think of dozens of libraries off the top of my head which are beautifully reusable and which have saved time and money that can never be calculated.

But to the extent that OOP has been a waste of time, I'd say it's because of lack of programmer training, compounded by the steep learning curve of learning a language specific OOP mapping. Some people "get" OOP and others never will.

Share

answered Aug 23, 2008 at 14:45

community wiki
user2189331

2 I just gave you an upvote that threw you above 2000 points - hopefully it will stick. :P – Jason Bunting Sep 8, 2008 at 23:52

5 It's rare to see OOP being taught effectively. – Jon W Sep 3, 2009 at 12:32

Your answer sounds a lot like one given by Agile/Scrum trainers when asked if it makes sense - "it's very useful if you do it right; if you fail then you must be doing it wrong". It's easy to throw around words like "reusable" and "maintainable", but it's neither easy to quantify these qualities in actual code, nor is there a recipe that tells you how to write good OOP (despite thousands of books attempting to do so). We also get the bad habit of ostensibly ignoring the hardware, which leads to horrible performance on the altar of avoiding premature optimization. – Tom Jan 28, 2016 at 15:13

24 votes

There's no empirical evidence that suggests that object orientation is a more natural way for people to think about the world. There's some work in the field of psychology of

programming that shows that OO is not somehow more fitting than other approaches.

> Object-oriented representations do not appear to be universally more usable or less usable.
>
> It is not enough to simply adopt OO methods and require developers to use such methods, because that might have a negative impact on developer productivity, as well as the quality of systems developed.

Which is from "On the Usability of OO Representations" from Communications of the ACM Oct. 2000. The articles mainly compares OO against theprocess-oriented approach. There's lots of study of how people who work with the OO method "think" (Int. J. of Human-Computer Studies 2001, issue 54, or Human-Computer Interaction 1995, vol. 10 has a whole theme on OO studies), and from what I read, there's nothing to indicate some kind of naturalness to the OO approach that makes it better suited than a more traditional procedural approach.

edited Jun 20, 2020 at 9:12

community wiki
3 revs, 3 users 80%
Svend

18 It works fine. What it can't do is force bad coders to write good code. There's a periodic hue and cry against it for that reason. – chaos Dec 31, 2008 at 4:43

6 @melaos: the summary is in the middle. OOP is one tool in the toolkit, not the only one, and there is no substitute for training, experience, and judgement. – Mike Dunlavey Dec 31, 2008 at 14:37

44 I find it kind of amusing when someone posts a "Question" to argue a point then accepts the first answer that supports his point, even though there are higher rated questions supporting the opposite. Human nature is cool. – Bill K Jul 1, 2009 at 0:50

3 @melaos, the summary (of those articles at least) is that nothing suggests OO to be a natural way of thinking for people, and thus, not inherently superior to any other way one might construct programs. – Svend Oct 20, 2009 at 15:58

6 @Bill K: It was one of the few answers that provided citations rather than hand-waving and mere insistence that OO "must" be better. If the referenced literature supports the notion that OO is not any particular improvement or of any particular benefit, and hence supports my original position, I'm not sure why I should disregard it. Can you provide similar references that counter my OP? – DrPizza Nov 30, 2009 at 6:46

21

votes

I think the use of opaque context objects (HANDLEs in Win32, FILE*s in C, to name two well-known examples--hell, HANDLEs live on the other side of the kernel-mode barrier, and it really doesn't get much more encapsulated than that) is

> found in procedural code too; I'm struggling to see how this is something particular to OOP.

`HANDLE`s (and the rest of the WinAPI) *is* OOP! C doesn't support OOP very well so there's no special syntax but that doesn't mean it doesn't use the same concepts. WinAPI is in every sense of the word an object-oriented framework.

See, this is the trouble with every single discussion involving OOP or alternative techniques: nobody is clear about the definition, everyone is talking about something else and thus no consensus can be reached. Seems like a waste of time to me.

Share

answered Aug 23, 2008 at 16:05

community wiki
Konrad Rudolph

1 I was just about to post something very similar. – Brad Gilbert Sep 15, 2008 at 16:17

I agree that you can use OOP concepts without special syntax, but on the other hand, I don't think that the WinAPI is a good example of OOP concepts. – Lena Schimmel Mar 6, 2009 at 1:55

14 votes

Its a programming paradigm.. Designed to make it easier for us mere mortals to break down a problem into smaller, workable pieces..

If you dont find it useful.. Don't use it, don't pay for training and be happy.

I on the other hand do find it useful, so I will :)

Share

answered Aug 23, 2008 at 15:14

community wiki
Rob Cooper

---

14 votes

Relative to straight procedural programming, the first fundamental tenet of OOP is the notion of information hiding and encapsulation. This idea leads to the notion of the ***class*** that seperates the interface from implementation. These are hugely important concepts and the basis for putting a framework in place to think about program design in a different way and better (I think) way. You can't really argue against those properties - there is no trade-off made and it is always a cleaner way to modulize things.

Other aspects of OOP including inheritance and polymorphism are important too, but as others have alluded to, those are commonly over used. ie: Sometimes people use inheritance and/or polymorphism because they can, not because they should have. They are powerful concepts and very useful, but need to be used wisely and are not automatic winning advantages of OOP.

Relative to re-use. I agree re-use is over sold for OOP. It is a possible side effect of well defined objects, typically of more primitive/generic classes and is a direct result of the encapsulation and information hiding concepts. It is potentially easier to be re-used because the interfaces of well defined classes are just simply clearer and somewhat self documenting.

Share

edited Aug 24, 2008 at 12:51

community wiki
2 revs
Tall Jeff

1 Re-use is something of a grail for software developers, isn't it, regardless of whatever paradigm they use, be it OO or functional or whatever? That comes into conflict with the ever-changing requirements made on software and the question seems to be one of making designs that are flexible.
– Geoglyph Jan 14, 2009 at 15:59

"notion of the class that seperates the interface from implementation" --- I thought interfaces were interfaces and classes were implementations? Maybe I'm just a too process-oriented thinker, but I figure it's the polymorphism, the variance in code with uniformity in use, that's the kicker of OOP; I figure that "a bunch of C functions" separate an interface from an implementation just as well as "a java class". Maybe I'm all wrong? – Jonas Kölker Jun 25, 2009 at 22:15

2 @Jonas - To your "a bunch of C functions" --- I agree that they can separate an interface from an implementation, and at that point it IS starting to be OOP. If the example continues on to define a C struct that the API operates on, you've basically

created encapsulation (especially if the API operates opaquely on the structure)...and then it really is OOP in C. – Tall Jeff Jun 26, 2009 at 0:38

12 votes

The problem with OOP is that it was oversold.

As Alan Kay originally conceived it, it was a great alternative to the prior practice of having raw data and all-global routines.

Then some management-consultant types latched onto it and sold it as the messiah of software, and lemming-like, academia and industry tumbled along after it.

Now they are lemming-like tumbling after other good ideas being oversold, such as functional programming.

So what would I do differently? Plenty, and I wrote a book on this. (It's out of print - I don't get a cent, but you can still get copies.)Amazon

My constructive answer is to look at programming not as a way of modeling things in the real world, but as a way of encoding requirements.

That is very different, and is based on information theory (at a level that anyone can understand). It says that programming can be looked at as a process of defining languages, and skill in doing so is essential for good programming.

It elevates the concept of domain-specific-languages (DSLs). It agrees emphatically with DRY (don't repeat yourself). It gives a big thumbs-up to code generation. It results in software with massively less data structure than is typical for modern applications.

It seeks to re-invigorate the idea that the way forward lies in inventiveness, and that even well-accepted ideas should be questioned.

Share

edited Dec 31, 2008 at 14:00

community wiki
2 revs
Mike Dunlavey

Cool. Since the book is out of print, you wouldn't want to provide PDFs, would ya? Amazon sends SLOWLY to Argentina... – Dan Rosenstark Dec 31, 2008 at 19:32

I suspect that somewhere early in the process, some good coders waxed rhapsodic about what they could do with OOP, and someone overheard them and thought that that meant *everyone* both could and would do likewise. – chaos Dec 31, 2008 at 19:44

@Daniel: good suggestion. I'll work on that, and see if I can attach it to my blogspot. You'll find it a little old-fashioned, 'cause it was in '94, but the principles haven't changed. – Mike Dunlavey Jan 1, 2009 at 15:35

1 @Mike Dunlavey Dear sir, may I support @yar's curiosity to know whether there's a chance to read / get your book [at least partially] via the internet? As it seems the way of effective

software [component] specification / implementation you propose / develop closely matches the one [I recently recognized] I would like to learn or be taught (as opposed to, well, the 'cargo' way introduced by nicely written but painfully ambiguous mainstream OO books). Thanks in advance [and cheers from Russia:)]. – mlvljr Apr 1, 2010 at 21:40

1 @mlvljr: OK. The quickest way might just be to scan it and make a big pdf file. I'll see if I can get to it in the next few days. Don't let me forget [sincere condolences for recent happenings in Moscow, and cheers from soggy Massachusetts].
– Mike Dunlavey Apr 1, 2010 at 22:36

11

votes

> HANDLEs (and the rest of the WinAPI) is OOP!
>
> Are they, though? They're not inheritable, they're certainly not substitutable, they lack well-defined classes... I think they fall a long way short of "OOP".

Have you ever created a window using WinAPI? Then you should know that you define a class (`RegisterClass`), create an instance of it (`CreateWindow`), call virtual methods (`WndProc`) and base-class methods (`DefWindowProc`) and so on. WinAPI even takes the nomenclature from SmallTalk OOP, calling the methods “messages” (Window Messages).

Handles may not be inheritable but then, there's `final` in Java. They don't lack a class, they *are* a placeholder for the class: That's what the word “handle” means. Looking at

architectures like MFC or .NET WinForms it's immediately obvious that except for the syntax, nothing much is different from the WinAPI.

Share

edited Jun 20, 2020 at 9:12

community wiki
Konrad Rudolph

The WINAPI is not OOP. It just shows a way of writing extendable procedural code. Good techniques are good no matter if they are OOP or not. I can write WINAPI programs in C and use the same techniques in my own code, so I guess C is OOP. – bruceatk Oct 10, 2008 at 12:38

3 It might not be what Alan Kay had in mind (google it!) but it's OOP nontheless. – Konrad Rudolph Oct 10, 2008 at 20:32

11 votes

Yes OOP did not solve all our problems, sorry about that. We are, however working on SOA which will solve all those problems.

Share

answered Sep 17, 2008 at 4:59

community wiki
Nat

4 Didn't you hear? SOA is the past. Today it's cloud computing that will be our saviour. – DrPizza Sep 17, 2008 at 9:59

I'm really wondering if your answers are ironical or not. I like irony, but usually its getting voted down, this makes me wonder... – Lena Schimmel Mar 6, 2009 at 2:03

I think this one IS ironic, and I won't vote it down. But I won't vote it up either! :-) – Yarik May 26, 2009 at 15:21

The question is fairly rhetorical so this is a valid answer (and the best answer). – Jeff Davis Jul 30, 2010 at 14:51

10

votes

OOP lends itself well to programming internal computer structures like GUI "widgets", where for example SelectList and TextBox may be subtypes of Item, which has common methods such as "move" and "resize".

The trouble is, 90% of us work in the world of business where we are working with business concepts such as Invoice, Employee, Job, Order. These **do not** lend themselves so well to OOP because the "objects" are more nebulous, subject to change according to business re-engineering and so on.

The worst case is where OO is enthusiastically applied to databases, including the egregious OO "enhancements" to SQL databases - which are rightly ignored except by database noobs who assume they must be the right way to do things because they are newer.

Share

answered Oct 23, 2008 at 16:12

community wiki

Tony Andrews

> That's all true, but... perhaps, the fact that OOP can simplify SOME business-unrelated aspects of an application (like UI implementation) is exactly one of the main reasons why a developer (finally!) can spend more time working on business-related aspects? – Yarik May 26, 2009 at 15:43

10

votes

In my experience of reviewing code and design of projects I have been through, the value of OOP is not fully realised because alot of developers have **not properly conceptualised the object-oriented model** in their minds. Thus they do not program with OO design, very often continuing to write top-down procedural code making the classes a pretty *flat* design. (if you can even call that "design" in the first place)

It is pretty scary to observe how little colleagues know about what an abstract class or interface are, let alone properly design an inheritance hierarchy to suit the business needs.

However, when good OO design is present, it is just sheer joy reading the code and seeing the code naturally fall into place into intuitive components/classes. I have always perceived system architecture and design like designing the various departments and staff jobs in a company - all are there to accomplish a certain piece of work in the grand scheme of things, emitting the synergy required to propel the organisation/system forward.

That, of course, is quite *rare* unfortunately. Like the ratio of beautifully-designed versus horrendously-designed physical objects in the world, the same can pretty much be said about software engineering and design. Having the good tools at one's disposal does not necessarily confer good practices and results.

Share

edited Dec 31, 2008 at 2:43

community wiki
3 revs
icelava

1 I've never achieved the sheer joy phase procedurally or OOP while reading code. :) – bruceatk Oct 10, 2008 at 12:29

1 Sheer joy, no, but a little smile perhaps? – Dan Rosenstark Dec 31, 2008 at 1:41

9
votes

Maybe a bonnet, lap or a tree is not a chair but they all are ISittable.

Share

answered Aug 25, 2008 at 20:19

community wiki
Johnno Nolan

2 You've never had to use an OO language without interfaces have you? You're lucky. – finnw Sep 28, 2008 at 19:27

No they aren't. They are things that were not designed for sitting, so I think to be true to the analogy, they would not have been written to implement the ISittable interface. They are from a 3rd party library, and now that you are writing a project involving a domain that includes sitting, you find you will have to write adapter objects to wrap them in to make them ISittable. See? Not very much like the real world. – EricS Feb 6, 2014 at 22:47

8

votes

> I think those real world things are objects

You do?

What methods does an invoice have? Oh, wait. It can't pay itself, it can't send itself, it can't compare itself with the items that the vendor actually delivered. It doesn't have any methods at all; it's totally inert and non-functional. It's a record type (a struct, if you prefer), not an object.

Likewise the other things you mention.

Just because something is real does not make it an object in the OO sense of the word. OO objects are a peculiar coupling of state and behaviour that can act of their own accord. That isn't something that's abundant in the real world.

Share

answered Oct 2, 2008 at 8:31

community wiki
DrPizza

2 Take any machine, electronic device, or living thing, those are all "couplings of state and behaviour". – TM. Nov 7, 2008 at 4:34

1 I agree that methods like Send() or Pay() are "unnatural" for invoices. But, for example, what about total amount as a derived but INHERENT attribute of an invoice? Isn't it an example of what OOP enables to model in a very "natural" way even for totally inert real-life entities? Not a huge achievement by itself, but still... – Yarik May 26, 2009 at 15:36

@Yarik: These "inert" entities lead to an Object-Based design. To me, the only proper OO is Simulation, where objects **"can act of their own accord"**, as this answer states. If OO is the only tractable way to accomplish something, fine, but otherwise can't we stick to something simpler and more like the problem being solved? – user4624979 Jul 1, 2015 at 14:56

7
votes

I have been writing OO code for the last 9 years or so. Other than using messaging, it's hard for me to imagine other approach. The main benefit I see totally in line with what CodingTheWheel said: modularisation. OO naturally leads me to construct my applications from modular components that have clean interfaces and clear responsibilities (i.e. loosely coupled, highly cohesive code with a clear separation of concerns).

I think where OO breaks down is when people create deeply nested class heirarchies. This can lead to complexity. However, factoring out common finctionality into a base class, then reusing that in other descendant classes is a deeply elegant thing, IMHO!

Share

answered Aug 23, 2008 at 15:02

community wiki
Sean Kearon

7
votes

In the first place, the observations are somewhat sloppy. I don't have any figures on software productivity, and have no good reason to believe it's not going up. Further, since there are many people who abuse OO, good use of OO would not necessarily cause a productivity improvement even if OO was the greatest thing since peanut butter. After all, an incompetent brain surgeon is likely to be worse than none at all, but a competent one can be invaluable.

That being said, OO is a different way of arranging things, attaching procedural code to data rather than having procedural code operate on data. This should be at least a small win by itself, since there are cases where the OO approach is more natural. There's nothing stopping anybody from writing a procedural API in C++, after all, and so the option of providing objects instead makes the language more versatile.

Further, there's something OO does very well: it allows old code to call new code automatically, with no changes. If I have code that manages things procedurally, and I add a new sort of thing that's similar but not identical to an earlier one, I have to change the procedural code. In an OO system, I inherit the functionality, change what I like, and the new code is automatically used due to polymorphism. This increases the locality of changes, and that is a Good Thing.

The downside is that good OO isn't free: it requires time and effort to learn it properly. Since it's a major buzzword, there's lots of people and products who do it badly, just for the sake of doing it. It's not easier to design a good class interface than a good procedural API, and there's all sorts of easy-to-make errors (like deep class hierarchies).

Think of it as a different sort of tool, not necessarily generally better. A hammer in addition to a screwdriver, say. Perhaps we will eventually get out of the practice of software engineering as knowing which wrench to use to hammer the screw in.

answered Dec 31, 2008 at 15:24

community wiki
David Thornley

I like your last paragraph the best. – Mike Dunlavey Dec 31, 2008 at 16:24

6

votes

@Sean

> However, factoring out common finctionality into a base class, then reusing that in other descendant classes is a deeply elegant thing, IMHO!

But "procedural" developers have been doing that for decades anyway. The syntax and terminology might differ, but the effect is identical. There is more to OOP than "reusing common functionality in a base class", and I might even go so far as to say that that is hard to describe as OOP at all; calling the same function from different bits of code is a technique as old as the subprocedure itself.

Share

answered Aug 23, 2008 at 15:08

community wiki
DrPizza

6

votes

@Konrad

> OOP may be flawed and it certainly is no silver bullet but it makes large-scale applications much simpler because it's a great way to reduce dependencies

That is the dogma. I am not seeing what makes OOP significantly better in this regard than procedural programming of old. Whenever I make a procedure call I am isolating myself from the specifics of the implementation.

Share

answered Aug 23, 2008 at 15:11

community wiki
DrPizza

6
votes

To me, there is a lot of value in the OOP syntax itself. Using objects that attempt to represent real things or data structures is often much more useful than trying to use a bunch of different flat (or "floating") functions to do the same thing with the same data. There is a certain natural "flow" to things with good OOP that just makes more sense to read, write, and maintain long term.

It doesn't necessarily matter that an Invoice isn't really an "object" with functions that it can perform itself - the object instance can exist just to perform functions on the data without having to know what type of data is actually there. The function "invoice.toJson()" can be called successfully without having to know what kind of data "invoice" is - the result will be Json, no matter it if comes from a database, XML, CSV, or even another JSON object. With procedural functions, you all the sudden have to know more about your data, and end up with functions like "xmlToJson()",

"csvToJson()", "dbToJson()", etc. It eventually becomes a complete mess and a HUGE headache if you ever change the underlying data type.

The point of OOP is to hide the actual implementation by abstracting it away. To achieve that goal, you must create a public interface. To make your job easier while creating that public interface and keep things DRY, you must use concepts like abstract classes, inheritance, polymorphism, and design patterns.

So to me, the real overriding goal of OOP is to make future code maintenance and changes easier. But even beyond that, it can really simplify things a lot when done correctly in ways that procedural code never could. It doesn't matter if it doesn't match the "real world" - programming with code is not interacting with real world objects anyways. OOP is just a tool that makes my job easier and faster - I'll go for that any day.

Share

answered Dec 31, 2008 at 4:22

community wiki
Vance Lucas

5

votes

@CodingTheWheel

> But to the extent that OOP has been a waste of time, I'd say it's because of lack of programmer

> training, compounded by the steep learning curve of learning a language specific OOP mapping. Some people "get" OOP and others never will.

I dunno if that's really surprising, though. I think that technically sound approaches (LSP being the obvious thing) make *hard to use*, but if we don't use such approaches it makes the code brittle and inextensible anyway (because we can no longer reason about it). And I think the counterintuitive results that OOP leads us to makes it unsurprising that people don't pick it up.

More significantly, since software is already fundamentally too hard for normal humans to write reliably and accurately, should we really be extolling a technique that is consistently taught poorly and appears hard to learn? If the benefits were clear-cut then it might be worth persevering in spite of the difficulty, but that doesn't seem to be the case.

Share

answered Aug 23, 2008 at 15:06

community wiki
DrPizza

> The "proper training" must be very long, abstract and complex these days. I know: I teach programming. Decades ago, many people could learn to do some kind of programming. I think that is no longer true. – user4624979 Jul 2, 2015 at 18:07

5

votes

@Jeff

> Relative to straight procedural programming, the first fundamental tenet of OOP is the notion of information hiding and encapsulation. This idea leads to the notion of the class that seperates the interface from implementation.

Which has the more hidden implementation: C++'s iostreams, or C's FILE*s?

I think the use of opaque context objects (HANDLEs in Win32, FILE*s in C, to name two well-known examples--hell, HANDLEs live on the other side of the kernel-mode barrier, and it really doesn't get much more encapsulated than that) is found in procedural code too; I'm struggling to see how this is something particular to OOP.

I suppose that may be a part of why I'm struggling to see the benefits: the parts that are obviously good are not specific to OOP, whereas the parts that are specific to OOP are not obviously good! (this is not to say that they are necessarily bad, but rather that I have not seen the evidence that they are widely-applicable and consistently beneficial).

Share

answered Aug 23, 2008 at 15:54

community wiki

DrPizza

5
votes

In the only dev blog I read, by that Joel-On-Software-Founder-of-SO guy, I read a long time ago that OO does not lead to productivity increases. Automatic memory management does. Cool. Who can deny the data?

I still believe that OO is to non-OO what programming with functions is to programming everything inline.

(And I should know, as I started with GWBasic.) When you refactor code to use functions, `variable2654` becomes `variable3` of the method you're in. Or, better yet, it's got a name that you can understand, **and if the function is short,** it's called `value` and that's sufficient for full comprehension.

When code with no functions becomes code with methods, you get to delete miles of code.

When you refactor code to be truly OO, `b`, `c`, `q`, and `Z` become `this`, `this`, `this` and `this`. And since I don't believe in using the `this` keyword, you get to delete miles of code. Actually, you get to do that even if you use `this`.

I do not think OO is natural metaphor.

I don't think language is a natural metaphor either, nor do I think that Fowler's "smells" are better than saying "this code tastes bad." That said, I think that OO is not about natural metaphors and people who think the **objects just pop out at you** are basically missing the point. **You define the object universe,** and better object universes result in code that is shorter, easier to understand, works better, or all of these (and some criteria I am forgetting). I think that people who use the customers/domain's natural objects as programming objects are missing the power to redefine the universe.

For instance, when you do an airline reservation system, what you call a reservation might not correspond to a legal/business reservation at all.

Some of the basic concepts are really cool tools

I think that most people exaggerate with that whole "when you have a hammer, they're all nails" thing. I think that the other side of the coin/mirror is just as true: when you have a gadget like polymorphism/inheritance, you begin to find uses where it fits like a glove/sock/contact-lens. The tools of OO are very powerful. Single-inheritance is, I think,

absolutely necessary for people not to get carried away, my own multi-inheritance software not withstanding.

What's the point of OOP?

I think it's a great way to handle an absolutely massive code base. I think it lets you organize and reorganize you code and gives you a language to do that in (beyond the programming language you're working in), and modularizes code in a pretty natural and easy-to-understand way.

OOP is destined to be misunderstood by the majority of developers

This is because it's an eye-opening process like life: you understand OO more and more with experience, and start avoiding certain patterns and employing others as you get wiser. One of the best examples is that you stop using inheritance for classes that you do not control, and prefer the *Facade* pattern instead.

Regarding your mini-essay/question

I did want to mention that you're right. Reusability is a pipe-dream, for the most part. Here's a quote from Anders Hejilsberg about that topic (brilliant) from here:

> If you ask beginning programmers to write a calendar control, they often think to themselves, "Oh, I'm going to write the world's best calendar control! It's going to be polymorphic with respect to the kind of calendar. It will have displayers, and mungers, and this, that, and the other." They need to ship a calendar application in two months. They put all this infrastructure into place in the control, and then spend two days writing a crappy calendar application on top of it. They'll think, "In the next version of the application, I'm going to do so much more."
>
> Once they start thinking about how they're actually going to implement all of these other concretizations of their abstract design, however, it turns out that their design is completely wrong. And now they've painted themself into a corner, and they have to throw the whole thing out. I have seen that over and over. I'm a strong believer in being minimalistic. Unless you actually are going to solve the general problem, don't try and put in place a framework for solving a specific one, because you don't know what that framework should look like.

Share

edited Dec 31, 2008 at 17:07

community wiki
6 revs
Yar

4

votes

> Have you ever created a window using WinAPI?

More times than I care to remember.

> Then you should know that you define a class (RegisterClass), create an instance of it (CreateWindow), call virtual methods (WndProc) and base-class methods (DefWindowProc) and so on. WinAPI even takes the nomenclature from SmallTalk OOP, calling the methods “messages” (Window Messages).

Then you'll also know that it does no message dispatch of its own, which is a big gaping void. It also has crappy subclassing.

> Handles may not be inheritable but then, there's final in Java. They don't lack a class, they are a placeholder for the class: That's what the word “handle” means. Looking at architectures like MFC or .NET WinForms it's immediately obvious that

> except for the syntax, nothing much is different from the WinAPI.

They're not inheritable either in interface or implementation, minimally substitutable, and they're not substantially different from what procedural coders have been doing since forever.

Is this really it? The best bits of OOP are just... traditional procedural code? *That's* the big deal?

Share

answered Aug 23, 2008 at 17:13

community wiki
DrPizza

You have to remember the Win32 interface was basically a reimplementation of Win16. Which was designed over two decades ago. – Brad Gilbert Sep 15, 2008 at 16:20

When I was learning to program in college, mostly in C but some other languages also, we were taught some basic ideas and exposed to a few platforms, but it was understood that the overall responsibility for coming up with designs, frameworks, best practices, etc. would be up to us in the workplace. We learned techniques, not worldviews. With OO, you have to learn a religion before you can do anything useful, and it completely determines how you see solutions. I don't think that is really proper. – user4624979 Jul 2, 2015 at 18:03

4

votes

I agree completely with InSciTek Jeff's answer, I'll just add the following refinements:

- Information hiding and encapsulation: Critical for any maintainable code. Can be done by being careful in any programming language, doesn't require OO features, but doing it will make your code slightly OO-like.
- Inheritance: There is one important application domain for which all those OO *is-a-kind-of* and *contains-a* relationships are a perfect fit: Graphical User Interfaces. If you try to build GUIs without OO language support, you will end up building OO-like features anyway, and it's harder and more error-prone without language support. Glade (recently) and X11 Xt (historically) for example.

Using OO features (especially deeply nested abstract hierarchies), when there is no point, is pointless. But for some application domains, there really is a point.

Share

edited May 23, 2017 at 12:25

community wiki
2 revs
Liudvikas Bukys

Yeah, I think you should use OOP for objects, functional programming for functions ... – Svante Nov 7, 2008 at 4:51

4

votes

I believe the most beneficial quality of OOP is data hiding/managing. However, there are a LOT of examples where OOP is misused and I think this is where the confusion comes in.

Just because you can make something into an object does not mean you should. However, if doing so will make your code more organized/easier to read then you definitely should.

A great practical example where OOP is very helpful is with a "product" class and objects that I use on our website. Since every page is a product, and every product has references to other products, it can get very confusing as to which product the data you have refers to. Is this "strURL" variable the link to the current page, or to the home page, or to the statistics page? Sure you could make all kinds of different variable that refer to the same information, but proCurrentPage->strURL, is much easier to understand (for a developer).

In addition, attaching functions to those pages is much cleaner. I can do proCurrentPage->CleanCache(); Followed by proDisplayItem->RenderPromo(); If I just called those functions and had it assume the current data was available, who knows what kind of evil would occur. Also, if I had to pass the correct variables into those functions, I am back to the problem of having all kinds of variables for the different products laying around.

Instead, using objects, all my product data and functions are nice and clean and easy to understand.

However. The big problem with OOP is when somebody believes that EVERYTHING should be OOP. This creates a lot of problems. I have 88 tables in my database. I only have about 6 classes, and maybe I should have about 10. I definitely don't need 88 classes. Most of the time directly accessing those tables is perfectly understandable in the circumstances I use it, and OOP would actually make it more difficult/tedious to get to the core functionality of what is occurring.

I believe a hybrid model of objects where useful and procedural where practical is the most effective method of coding. It's a shame we have all these religious wars where people advocate using one method at the expense of the others. They are both good, and they both have their place. Most of the time, there are uses for both methods in every larger project (In some smaller projects, a single object, or a few procedures may be all that you need).

Share

answered Sep 15, 2008 at 21:27

community wiki
Jeff Davis

---

3 votes

I don't care for reuse as much as I do for readability. The latter means your code is easier to change. That alone is

worth in gold in the craft of building software.

And OO is a pretty damn effective way to make your programs readable. Reuse or no reuse.

Share

answered Dec 31, 2008 at 13:30

community wiki
G S

---

2 votes

"The real world isn't "OO"."

Really? My world is full of objects. I'm using one now. I think that having software "objects" model the real objects might not be such a bad thing.

OO designs for conceptual things (like Windows, not real world windows, but the display panels on my computer monitor) often leave a lot to be desired. But for real world things like invoices, shipping orders, insurance claims and what-not, I think those real world things are objects. I have a stack on my desk, so they must be real.

Share

answered Oct 1, 2008 at 8:08

community wiki
S.Lott

2

votes

The point of OOP is to give the programmer another means for describing and communicating a solution to a problem in code to machines and people. The most important part of that is the communication to people. OOP allows the programmer to declare what they mean in the code through rules that are enforced in the OO language.

Contrary to many arguments on this topic, OOP and OO concepts are pervasive throughout all code including code in non-OOP languages such as C. Many advanced non-OO programmers will approximate the features of objects even in non-OO languages.

Having OO built into the language merely gives the programmer another means of expression.

The biggest part to writing code is not communication with the machine, that part is easy, the biggest part is communication with human programmers.

Share

answered Dec 17, 2008 at 23:12

community wiki
Bernard Igiri