

## Re: dip Notions 2 Major Errors

**Source:** <http://coding.derkeiler.com/Archive/General/comp.object/2004-09/1178.html>

---

**From:** Mark Nicholls ([nicholls.mark\\_at\\_mtvne.com](mailto:nicholls.mark_at_mtvne.com))

**Date:** 09/21/04

Date: Tue, 21 Sep 2004 16:07:13 +0100

"Ilja Preuß" <[it@iljapreuss.de](mailto:it@iljapreuss.de)> wrote in message  
news:4150106c\$1@news.totallyobjects.com...

> *Mark Nicholls wrote:*

> > *It is an observation about one value of subtyping \*and\* of different  
> > techniques that have a similar effect (introducing an abstraction  
> > between client and server).*

> >

> > *OK, indirection in general, if we restrict to subtyping we wont get an  
> > explosion of examples.*

>

> *OK.*

>

> > *But the two are still coupled. To get your binary components, you  
> > need to compile their source code. If the source code of component A  
> > depends on the source code of component B, every time you change B  
> > you need to recompile the source code of \*both\* and are likely to  
> > need to redeploy \*both\* binary components.*

> >>

> >

> > *hmmmm, this does not seem to be true to me.*

> >

> > *components are independenly deployable....and I can version each as I  
> > choose. if C---->A....I can change A and redeploy just A....given  
> > that I don't breach the contract.*

> >

> > *I did this with dll's, COM and .net.*

>

> *You are right, of course, that this is possible as long as you are very  
> attentive and/or have the right tool support. But I haven't yet seen  
perfect*

> *tool support for this – even the very good incremental Eclipse Java  
compiler*

> *from time to time decides to recompile all the 80+ projects in my  
workspace*

> *when I changed an implementation detail in a low level module that  
shouldn't*

> *have any impact on the clients. Applying DIP to those modules can actually*

comp.object: Re: dip Notions 2 Major Errors

> *reduce compile time from several minutes to just an unnoticable moment.*

Again though, if the interfaces are in their own package, you can alter client, add new client, alter implementation, add implementation (though you need to instantiate it somewhere), with minimal fuss, you only need to recompile the package that contains the class, unless you alter contract i.e. interface or constructor.

Putting the interface in the client package potentially limits your options.

Compilation times, have never been top of my list of problem, but maybe my apps are more noddy.....:-)

or my tools better.

>  
>> *I accept you need to talk about both, and both are intrinsically*  
>> *linked, but we can draw a class diagram and make statements about the*  
>> *dependencies in that diagram, and then create scenarios about which*  
>> *class is deployed where we can make statements about the dependencies*  
>> *in that diagram*  
>  
> *Yes...*  
>  
>> *...given the class diagram is fixed.....seperation of*  
>> *fixed from variable...i.e. experimental constraints....to allow*  
>> *everything to change independently make s any hard conclusions*  
>> *virtually impossible to make.*  
>  
> *I didn' understand this part at all, sorry. Could you please try to*  
> *rephrase?*

OK, we do the design/analysis...and create class diagram that is based on the problem we want to solve.

We can talk about explicit seperating a class and interface (and operation from instantiation) and see new dependencies or observe inversions (lets not go there).

We can give scenario's given a fixed class model on where we can assign classes to packages and observe circular dependencies and/or genuine inversions (subject to labelling).

i.e. we can do thought experiments on class/package allocations and keep the class structure the same and make observations about different experiments.

>  
>>>>>> *I agree, but that's not the point of DIP. The point of DIP is*  
>>>>>> *that the concrete A now \*depends\* instead of being depended upon*  
>>>>>> *by it's callers.*  
>>>>>>

> >>>>> *logical static again....*  
> >>>>  
> >>>> *The important point is that, while now doing the depending, A still*  
> >>>> *\*receives\* messages.*  
> >>>>  
> >>>> *And is also depended on by the instantiating client, do not loose*  
> >>>> *this dependency...it has moved, not gone away or been inverted.*  
> >>>>  
> >>>> *It has gone away from the original client.*  
> >>>>  
> >>>> *This is too weak for me.*  
> >>>>  
> >>>> *after the mapping I have demonstrated, I believe it has not.*  
> >>>>  
> >>>> *It is in a different class to that, that operates via the interface.*  
> >>>>  
> >>>> *It may be in a different package from that that operates via the*  
> >>>> *interface.*  
> >>>>  
> >>>> *Yes. In many situations, that's a big, important improvement.*

yes.

but the client class has now been decomposed into at least two classes, to talk about the 'client' and identify only one is dangerous as we loose the instantiation behaviour, which we must keep.

the class has to be instantiated, and it may be done from the same package or a different one, from the same 'application'/process or a different one...but it has to be instantiated.

>  
> >>> *That's the important part – the*  
> >>> *high level implementation of the client doesn't care any longer*  
> >>> *about what low level service implementation it is using, nor where*  
> >>> *it is coming from. It might not even be decided upon by code I*  
> >>> *write; for example when using JAXP, to get a SAX parser I just call*  
> >>> *a method on a factory – which implementation I get depends on what*  
> >>> *features I requested and what implementations are available at*  
> >>> *runtime (which jar files are in the classpath).*  
> >>>>  
> >>>> *But the need to be available....so there is a logical dependency.....*  
> >>>>  
> >>>> *There is no compile time package dependency...but there is a run time*  
> >>>> *package dependency.*  
> >>>>  
> >>>> *But not on a \*specific\* implementation. The client (that is the high level*  
> >>>> *rules) can actually be reused with different low level implementations.*

well here we may diverge...if there is no dependency then if the package is not available it wont work. If you rename the class, the application wont work, fi you change the instantiation behaviour, i.e. the constuctor signature, the application wont work.

So there is a dependency...but it is not enforced by a compiler or linker, we wait for run time and cross our fingers....which is sometimes quite sensible.

>  
> >> *In fact this is what the DIP is about: inserting a point of  
> >> indirection, the abstraction, between high level client and low  
> >> level server. For DIP it's not important wether you use subtyping or  
> >> something else to do that.*  
> >  
> > *OK, my irritation with this is simply it's a declaration of  
> > indirection....which is why I believe Elliott keeps refering to in  
> > terms of the patterns that support this.*  
> >  
> > *Subtyping is the most common form of this in OO, but I don't believe  
> > we need to wrap it up in DIP, it obscures the the real message, and I  
> > believe implies incorrect consequences.*  
>  
> *What, for you, is "the real message of subtyping"???*

oh no.....that's put me on the back foot.

Booch/Rumbaugh/Gamma/Wirfs-Brock have failed me for a sales pitch for subtyping...so I'll have to splodge together some clumsy sentence, that will be wripped from the page, by those that know better.

I don't believe it has much or anything to do about package dependency or inversions or even versioning....and I think this obscures the message....

Subtyping, to me, is the ability to seperate implementation from interface and instantiation from operation.....though I believe they are always seperatae concepts, but are overloaded by name in most OOP's and UML, which is unfortunate.

thus we can vary both implementation and operator...subject to them satisfying some hopefully formal notion of substitutability.

note both...implementation and implementor, not just one.....to me the abstraction in general is 'owned' by both the set of client operators and the set of implementors.....not just one set or the other.

This is a pretty ugly statement so far, but....it allows us to abstract both client operation implementation (instantiation is harder and done via factories) from class implementation and implementation from client operation by inserting a layer of polymorphic indirection.

both client and class implementation....equally

and it stems from the context of the problem domain, the problem drives the solution.....

You could write a book about it....oh lots of people have....Rumbaugh/Booch etc....some we agree with and some not.

I will now get whipped to shreds by a pack of baying hounds....I expect Elliott can and will supply a quote that genuinely illustrates the point much better.

>  
> >> *Obviously you don't agree on the "inversion" part, but I don't think*  
> >> *that's that important.*  
> >  
> > *Probably not, we can agree to disagree on this, I wont loose any*  
> > *sleep, and I suspect neither will you.*  
>  
> *Correct! :)*  
>  
> >>>> *Does that compute to you?*  
> >>>>  
> >>>>  
> >>>> *I wish you wouldn't say that....I can never work out whether it is*  
> >>>> *an insult or a joke.*  
> >>>  
> >> *Sorry, I didn't mean it either way – I wasn't aware of the fact that*  
> >> *it would typically be interpreted that way... :o*  
> >  
> > *I usually take it as a joke, I am English, and it could be construed*  
> > *as a 'are you stupid' or a play on the word 'compute'.*  
> >  
> > *If this group does anything for me, it at least shows me how language,*  
> > *especially between different nationalities can be interpreted in*  
> > *different ways.*  
>  
> *Well, it's probably more an artefact of my imperfect mastery of the*  
*english*  
> *language... ;)*

Actually the jump between US and Canadian English to English English (all nations claim fluency) is often harder than from other cultures.

Writing things rather than talking tends to magnify any nuance into a potential insult/joke...friendly sarcasm, a favourite of the British often offends others.

>  
> *Regards, Ilja*  
>

comp.object: Re: dip Notions 2 Major Errors

>  
Regards also.