

Re: OODesign – OPF, design pattern

Source:

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.non-technical/2007-08/msg03172.html>

- *From:* "Joanna Carter [TeamB]" <joanna@xxxxxxxxxxxxxx>
 - *Date:* Tue, 28 Aug 2007 17:36:24 +0100
-

Aleksander Oven a écrit :

But it's after reading posts like this one from Peter, when the only thing I can say to myself is:

O..M..G!

How on earth is this simplifying application development?

In Peter's world, it is simple but then, some of the things he mentioned sound fairly experimental to me :-)

Sure, using Observer pattern here and there can really be useful. A couple of other design patterns are great, too. And I simply couldn't live without using interfaces to decouple my objects. But designing the whole application around such decoupled layers? That's just insane!

<grin> <evil cackle>

Let me explain...

A few years back, I tried to implement a small application this way. What constantly kept getting in my way was the fact, that UI controls and the RTL and VCL itselfs simply aren't built to support MVP/MVC-style of development. I couldn't use any of the more advanced components out there, because they were doing too many internal magic that wasn't exposed to my controller layer. Just handling drag-n-drop was almost impossible. And responding to dynamic user input? Forget it! Tracking the focus changes, intercepting messages, etc. It got really ugly really fast. I ended up having a whole lot of abstraction leakage between the layers and was finding myself constantly having to resort to hacks to minimize those leaks.

This lead me to believe that MVP/MVC might still be possible in theory,

Re: OODesign – OPF, design pattern

but with one important sacrifice: you must either stick to the simplest UI controls and forget about the rich 3rd-party components, or be prepared to write a whole lot of wrappers and even modify the source code of many of the controls in order get them to bend to your will.

And as I've gathered from OO-experts' posts on numerous occasions, this is exactly what it takes – writing wrappers and reinventing. I guess it's not unusual to see OO-experts writing their own TEdit and TButton implementations.

In fact, what it takes is to write quite a small hierarchy of "adapter" classes to allow you to use whatever components you want.

We are using the DevEx .NET components including the sophisticated XtraGrid and we are able to hook up our list of objects using an Interactor designed for the purpose. In fact, depending on the exact derived type of Interactor, we get differing grid behaviour without altering a line of code in the grid or the form.

However, we have written our own derived XtraGrid component but only to allow us to use our own customised design-time editors.

And that's the insane part, IMHO. By developing this separate (and yet strangely tight-coupled) framework you sometimes double and even triple the amount of code that needs to be written in order to achieve even the simplest of tasks. And this is not so innocent when it comes to maintaining all this code.

The truth is, that once you have written an adapter framework for your favourite widget set, you have written it once and for all; it gets used over and over again. Having been tested thoroughly, the framework will need very little alteration or maintenance.

I haven't yet seen an OO-based* application, that wasn't at least twice as hefty than its plain counterpart.

If you count the frameworks as part of the application then, yes such apps will seem hefty. But if you realise that the framework is not intended to be used just for the one application, and that it can be used for many more apps, without modification, then the app shrinks in size as the framework is considered to be external to it.

* This is actually one of my pet peeves. OO used to mean "Object Oriented", but nowadays it's more a synonym for "Object Persistence Framework Oriented". By this definition you could say that most developers out there (including me) aren't writing OO code. Which is kind of insulting, when I think of it.

Re: OODesign – OPF, design pattern

OPFs are but a small part of OO design. Their purpose is to separate out database differences from the object model; that is all, period. Once written, they fade into the background.

Saying that most developers are not writing OO code is not an insult, it is a simple statement of fact that most jobs get pushed into existence by forces outside the developers' control using a RAD, "OOish" IDE where everything gets slammed on the form because "that's the way it works" :-)

I have first hand experience in what it takes to debug this kind of applications. The fact that I've designed it myself doesn't help.

This is where you have to be very disciplined and do your testing before you integrate low-level stuff too deeply. I constantly have battles with clients who don't want to take the time to test classes before integrating them with the next level of complexity.

The design started as extremely elegant (don't they always?). All business classes were neatly isolated from the UI, with interfaces published in separate units as to minimize inter-class dependencies. There were observers, subjects, aspects, factories, actors, visitors, chains of command – the whole shabang. All in all, a really nice class framework. I can't help but still be a little proud of it even today.

Justifiably proud.

But the pride soon fades when I remember how it started growing in complexity as I added features. In mature development stages, there are often some features that weren't anticipated from the start. And this is where things start to really fall apart. I've seen this happen to others, too! It's one of the reasons why I'm presently against overuse of such frameworks – they can quickly turn into ball-n'-chain if you don't anticipate the unanticipated features. You become locked in by your own clever design!

It's called feature creep and has to be monitored very carefully.

Soon, there's a whole lot of notifications flying around all the time,

<snip>

It turns out infinite notification loops are all too common when complexity grows. And there's no definite counter-measures to fight them, except for iron discipline during development, and – you guessed it – more code! Checks on every critical entry-point to make sure we

Re: OODesign – OPF, design pattern

don't end up in a notification cycle of doom.

From a quick thought about this kind of problem, I would say that it is a case, not of overuse of frameworks, but maybe, overuse of notifications :-)

We don't use the Observer pattern that much; its main purpose in our frameworks is to keep UI controls up to date with business models, although we do use it in limited other places.

Seriously, though... This all invariably causes a lot of stress – it's really hard having to concentrate on so many intricate details almost all the time. One minute you might be submerged deep into writing business logic and then the next, you'll be walking up the seemingly infinite call stack, trying to figure out why that darned label isn't getting updated with the new caption.

IMO, this violates one of the key principles of software development – reducing the amount of context switching in your brain. If anything, this kind of development increases and intensifies those switches! And I'm sure even experts get annoyed by this sometimes.

Which is why you need to develop *and* test classes before integration, preferably using unit testing to allow you to easily check for regression if you do have to add an unforeseen feature. One unit test can save days of heartache :-)

I can't help but feel that this extreme OO is just one of the many variations on the same theme in software industry – emperor's (relatively) new clothes.

Except that OO is not new, it has been around for decades in one form or another. IMO, there is no such thing as extreme OO, the .NET framework is designed with everything being an object and is a stunning example of a fairly well designed framework that stands up to reuse very well indeed.

I can't even start to imagine what extra development and maintenance costs such design infers on an moderate to fairly complex project. It seems mind-boggling that someone would embark on a 2+ year long journey of software design and development with serious intentions to build it entirely on OPF principles. I imagine it would never reach production stage.

I have been working with my present client for somewhere around a year of effort (spread out over three years) on the latest iteration of my frameworks; but that does include a month or so of trying to get things to work in Delphi for Win32 before realising that we just had to move to ..NET to get the degree of reflection and functionality required to support our ideas.

Re: OODesign – OPF, design pattern

I wonder how much time it would take me to if I'd have written it using
OO principles in the first place? :D

<vbg>

Joanna

--

Joanna Carter [TeamB]
Consultant Software Engineer

.