Should you really re-invent the wheel?

It's often debated whether or not as a website developer / designer, that you should or should not re-invent the wheel. For clarification – this usually implies, anything such as frameworks or scripts that have already been commercially or non commercially developed to a considerable level already.

What you will learn...

- Further in-depth knowledge about OO & Frameworks

What you should know...

- A good understanding of PHP
- Understanding of OO Programming Paradigm

There are many leading developers on the forefront of the market who share this view, but are also willing to waiver it depending on various circumstances. Andrew Embler, CTO of Concrete5, is such a man. A few questions were posed towards Andrew and his success story behind Concrete5. To get a real understanding of what *re-inventing* the wheel is and when is it really a valid excuse, we'll be taking a look at Andrews response in the brief interview...

What prompted you to get involved with the creation of Concrete 5 and what would you say MVC (model view controller) has to offer over other models in PHP?

We started working on what would ultimately become concrete5 in late 2003. As with many things on the web, this CMS actually was born out of a real need for a real project – in this case a Lewis and Clark bicentennial site for the Ad Council. I was the tech lead on the project, and worked with Franz (concrete5's CEO) to implement what we knew we needed for the site. This included an early version of our in-context editing interface. Between 2003 and 2008, we improved upon Concrete CMS and kept it an internal, commercial CMS for use with our clients. In 2008, I started working internally on a nearly complete rewrite for Concrete version 5.0. This project became known as concrete5, and we thought the code had so much potential that we decided to try and get it used as widely as possible, and so we released it open source in the fall of that year.

As far as the model-view-controller pattern goes, Concrete CMS initially didn't really adhere to MVC at all. We started as most PHP projects start: quickly, without a tremendous amount of initial design (beyond the concept of blocks and collections/pages – which still remain pretty unchanged from our original version 1.0.). During the interim years between 1.0 and the complete rewrite of concrete5, I had the opportunity to work with frameworks like CakePHP and Code Igniter, and their implementations of MVC really appealed to me. Going back to Concrete CMS without those pieces available felt really limiting, and when it came time for version 5.0, I knew I wanted some of those MVC concepts to make their way into concrete5.

Tackling the 'real' parts of PHP like any language can be tricky enough, so for the novices out there, if you had to choose a framework what would it be and why?

I'd say concrete5 – but then again I might be a bit biased. In all seriousness, if you're tackling a website that needs some real content management and in-browser editing, I think concrete5 is a great choice, especially when you can't necessarily see how the site might grow and mature in the future. concrete5 strikes a nice balance between being easy to edit with and being useful and extremely extendable for developers who want to make any type of website.

If what you're building is more of a web application than a website – and I realize the distinction is a fuzzy and sometimes arbitrary one – and perhaps has less need for content management, I'd recommend Symfony 2 or Zend Framework. We actually use a number of Zend Framework libraries in concrete5, and couldn't do a lot of what we do if it weren't for that project.

Before the creation of *Concrete 5*. What was your biggest gripe (if you don't mind saying) about the state of PHP frameworks?

For me, frameworks didn't quite go far enough. This is a matter of opinion, and I imagine there are those out there who'd violently disagree with me, but I always found it a shame when frameworks didn't support the concepts of authentication, user objects, extendable data types/attributes, along with the interfaces to support them. I always disliked the fact that I had to code up so much of the UI when working with PHP frameworks. Fortunately, with great UI libraries like TwitterBootstrap now available, it's easier than ever to make interfaces that look great for custom web applications.

Playing a bit of devils advocate here, I've always sided with the argument - *don't re-invent the wheel,* what do you disagree or agree about this when in the context of creating a bespoke CMS, Framework vs an open-source framework?

That's a really good question. Generally I'd agree with this, but sometimes you just feel like you can build a better solution to a problem you see people having, even though there are other systems that tackle the same or similar problems. If that's the case, I say don't be afraid to try it, even if you do run the risk of reinventing the wheel. When we started with Concrete CMS we didn't see any PHP content management systems that offered in-context editing the way we were planning to. As Concrete matured, some systems have added this capability in varying ways, but we've really benefited from having our in-context interface be at the forefront of every decision we've ever *made.*

You do raise a good question about existing MVC frameworks and concrete5. In early 2008, when I first started the full rewrite of what would become concrete5, I was initially convinced that the rewrite would be less wholesale than it ultimately became. To that end, I thought that bolting on an existing, external open source framework like Symfony or CakePHP would be too difficult and time-consuming. Ultimately, however, we did rewrite most of the entire application, but at that point the ship had sailed regarding whether we would use custom MVC functionality or an existing framework. I don't regret the decision, since using our own MVC framework has let us do some really interesting things with blocks, views and controllers, but it does mean that people have to learn a new way of developing when they pick up concrete5. Judging by our developer community, once they learn the concepts they don't really mind; in fact, they're pretty excited.

Frameworks are a great way in which one can learn new methods through forced industry practices. Although there are cases in the UK alone - where an aptitude to develop web applications in Zend for instance, is desirable. Is it truly beneficial when starting out, to implement the use of such frameworks?

This purely comes down to opinion. However, there are some facts that linger. Learning a framework, especially when one is not necessarily adept in all the intricacies of the language being used – can be time consuming and costly (if you're running a business or work as a freelancer). It is often desirable to get the work done without having to learn on the job as this proves for a more efficient meeting of deadlines and smoother work flow.

Of course, there will be many occasions when learning on the job occurs, but you have to ask yourself. Is this going to help you progress and gain a good rapport with some of your clients, or is this going to cause massive delays within the project deadlines?

On the flip-side however, frameworks can dramatically reduce the number of hours it would take to develop a web based application by providing an abstraction layer for the databases and by providing many methods and functionality to menial tasks you would otherwise

Sometimes you just feel like you can build a better solution to a problem you see people having
– Andrew Embler.

have to implement yourself. A framework if used properly (also, if the user understands it) can be a life saver as well as a great development tool in any freelancers arsenal.

What about myself?

Having seen the many frameworks out there ranging from Zend to Cake PHP. I have taken a similar approach to everything stated above. The Zend framework is the weapon of choice when developing a web application which is going to be vast and complex.

So, what made me decide to create my own framework?

Well, this is rather simple, about a year ago I was working on a very large project to do with sustainability, the pay was decent & the royalties plenty. However, after finishing the project. A question still lingered at the back of my mind. *What other way could it have been implemented, that would be better and more efficient?*.

After a short period of time I started to sketch out and develop a non MVC structured framework with the name 'Argent Framework' shortened down to AF. This has the bog standard (and I mean, very bog standard) methods which you'd expect from most frameworks. But, my goal was to be able to attach *blocks* visually and have the framework create the file structure for me. At first this wasn't the case, and it wasn't until I took a long break and implemented this functionality just a few months back that this came to fruition.

The core concepts works off of using Processing.js (which is a Java library) as the UI (user interface) this communicates with the website - which then communicates with the database hierarchy. This framework seems very well suited towards either, small websites or single purpose data systems. Although this is a relatively niche area. I have aspirations of taking the framework further. Possibly, a rewrite in Ruby as well.

What about the API?

There are literally only a few methods within AF, for instance.

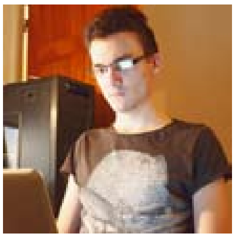
```
$this->retrob('processing')->setCanvas(500,500,'square.
                    pcs');
```

This sets the canvas size in HTML5 for the processing window and loads the processing sketch in question.

From a developers point of view, however; this is great as you can position this anywhere in the site and expect the canvas to appear. From a designers point of view, that may be complicated. Thus, they can use shorthand calls to this method like so (within their template file).

```
[@processing-setCanvas:500,500,square.pcs]
```

If you are interested in reading up a more in depth version of what it is, AF is about. Please feel free to check it out at
www.argentgray.com/whatisaf



Article prepared by Michael Gray