

Monday, February 27, 2006

The no-framework PHP MVC framework

March 1, 2006 - Disclaimer: Since a lot of people seem to me misunderstanding this article. It isn't about OOP vs. Procedural programming styles. I happen to lean more towards procedural, but could easily have gone more OOP. I simplified the code a bit for brevity, but have added a light OO layer back in the model now. Not that it makes a difference. What I was hoping to get across here is a simple example of how you can use PHP as-is, without additional complex external layers, to apply an MVC approach with clean and simple views and still have all the goodness of fancy Web 2.0 features. If you think I am out to personally offend you and your favourite framework, then you have the wrong idea. I just happen find most of them too complex for my needs and this is a proposed alternative. If you have found a framework that works for you, great.

So you want to build the next fancy Web 2.0 site? You'll need some gear. Most likely in the form of a big complex MVC framework with plenty of layers that abstracts away your database, your HTML, your Javascript and in the end your application itself. If it is a really good framework it will provide a dozen things you'll never need.

I am obviously not a fan of such frameworks. I like stuff I can understand in an instant. Both because it lets me be productive right away and because 6 months from now when I come back to fix something, again I will only need an instant to figure out what is going on. So, here is my current approach to building rich web applications. The main pieces are:

PHP 5
Yahoo! User Interface Library
JSON

MVC?

I don't have much of a problem with MVC itself. It's the framework baggage that usually comes along with it that I avoid. Parts of frameworks can be useful as long as you can separate the parts out that you need. As for MVC, if you use it carefully, it can be useful in a web application. Just make sure you avoid the temptation of creating a single monolithic controller. A web application by its very nature is a series of small discrete requests. If you send all of your requests through a single controller on a single machine you have just defeated this very important architecture. Discreteness gives you scalability and modularity. You can break large problems up into a series of very small and modular solutions and you can deploy these across as many servers as you like. You need to tie them together to some extent most likely through some backend datastore, but keep them as separate as possible. This means you want your views and controllers very close to each other and you want to keep your controllers as small as possible.

Goals for this approach

Clean and simple design

HTML should look like HTML

Keep the PHP code in the views extremely simple: function calls, simple loops and variable substitutions should be all you need

Secure

Input validation using `pecl/filter` as a data firewall

When possible, avoid layers and other complexities to make code easier to audit

Fast

Avoid `include_once` and `require_once`

Use APC and `apc_store/apc_fetch` for caching data that rarely changes

Stay with procedural style unless something is truly an object

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Avoid locks at all costs

Example Application

Here is the example application I will be describing.

It is a form entry page with a bit of Javascript magic along with an sqlite backend. Click around a bit. Try to add an entry, then modify it. You will see the server->client JSON traffic displayed at the bottom for debug purposes.

The Code

This is the code layout. It uses AJAX (with JSON instead of XML over the wire) for data validation. It also uses a couple of components from the Yahoo! user interface library and PHP's PDO mechanism in the model.

The presentation layer is above the line and the business logic below. In this simple example I have just one view, represented by the add.html file. It is actually called add.php on the live server, but I was too lazy to update the diagram and it really doesn't matter. The controller for that view is called add_c.inc. I tend to name files that the user loads directly as something.html or something.php and included files as something.inc. The rest of the files in the presentation layer are common files that all views in my application would share.

ui.inc has the common user interface components, common.js contains Javascript helper functions that mostly call into the presentation platform libraries, and styles.css provides the stylesheet.

A common db.inc file implements the model. I tend to use separate include files for each table in my database. In this case there is a just single table called "items", so I have a single items.inc file.

Input Filtering

You will notice a distinct lack of input filtering yet if you try to inject any sort of XSS it won't work. This is because I am using the pecl/filter extension to automatically sanitize all user data for me.

View - add.html

Let's start with the View in add.html:

The main thing to note here is that the majority of this file is very basic HTML. No styles, or javascript and no complicated PHP. It contains only simple presentation-level PHP logic. A modulus operation toggles the colours for the rows of items, and a loop around a heredoc (

Posted by Rasmus in PHP at 14:39

It is downright scary how much we think alike. Even your code looks so much like mine you'd think I stole everything I do from you--which in a way is partially true. Great article, I never understood why so many programmers want to force layer after layer of complexity onto a simple Web application.

On a side note, I'm curious to see how developers will take to Yahoo! REST requests being returned as serialized PHP.
Anonymous on Feb 27 2006, 23:07

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

But there are so many things that you find yourself doing over and over again that you start to make sense of multilayered frameworks. They don't have to be necessarily complex, but I find extremely useful to work with simple interfaces to objects that do trivial things - things that I don't really need to reimplement. I can always override the default model or controller if I want more complex functionality...Caching takes care of that extra performance hit. I've made my own multi layered framework and up until now it's made me happy, and I don't see why I shouldn't use some other people's framework if it is well designed and solves the common problems I don't need to go over - again.
Anonymous on Feb 28 2006, 07:57

I think you are confusing component re-use with frameworks here. If you always build exactly the same style of application and you have a framework tuned perfectly for that task, then you are in luck. Finding a "well-designed" framework as you say, that fits your problem space perfectly is easier said than done. My contention is that your time is better spent building your own ideal base infrastructure for any sizable project and then use whatever discrete components you are used to on top of that.
Anonymous on Feb 28 2006, 14:31

That's all great if all you need to do is build the web's next best guestbook, but if you work on a project with 7 developers and 1 webdesigner this is not going to work very well.

A framework isn't ment to just add layers of complexity, it also enforces a certain way of coding (ok I realize you could just as well write coding specs).

Your statement about keeping it procedural as much as possible means losing ALL the benefits of OOP in the first place.

Next please compare your controller to (for example) this:

```
class ItemController
{
function ItemController()
{
$this->model = new ItemModel();
$this->view = new ItemView($this->model);
}

function run()
{
$action = isset($_POST['itemAction']) ? $_POST['itemAction'] : 'listItems';
switch($action)
{
case 'listItems':
$this->view->renderList();
break;

case 'addItem':
$this->model->addItem();
$this->view->renderList();
break;
}
}

$controller = new ItemController();
$controller->run();
```

Which one do you find easier to come back to after 1 year?

All that's left to say is : globals? SQL Injection?
I can't believe what you posted, I'm downright shocked to the bones.
Anonymous on Mar 1 2006, 04:40

Which one do you find easier to come back to after 1 year?

All that's left to say is : globals? SQL Injection?
I can't believe what you posted, I'm downright shocked to the bones.

This is very effective and maintainable. In fact, I think it'd be even more maintainable than some object oriented mess of code, especially when that object oriented mess of code, is just a mess.
Anonymous on Mar 1 2006, 06:37

I have had great luck with <http://phpsavant.com> which I would characterize as the anti-template templating system. No special syntax required, just create a new template, assign data to it from your controller, and display it. The way you access assigned data from the template is with the object prefix "\$this->" instead of any new special syntax.
Anonymous on Mar 1 2006, 06:52

Rasmus this really isn't meant to be a flame but I'm dumbstruck. That is spaghetti code you're advocating pure and simple.

In the anarchic and often anti-intellectual world of php there needs to be some leadership. Anyone who seriously wants to learn how to program is badly served by articles such as above. Most people churn out code exactly like this but there must be clear goals for

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

those who want to go on to learn more - OOP & testing basically.

I don't doubt that you found some bad examples of OOP frameworks. That's not surprising given the lack of an object culture (<http://www.procata.com/blog/archives/2006/01/13/building-a-culture-of-objects-in-php/>) in php but it doesn't mean that OOP is a bad choice. OOP doesn't add complexity: it's a way to tame the complexity which already exists but which might not have been understood.

Anonymous on Mar 1 2006, 07:23

I fully agree with You Jurgen,

@Andrew: We assume that OO and procedural code is correct of course... So no mess..

This way of working is excellent for small websites with just one maintainer, and maybe one developer sitting close to each other, simple as that.

But when things start to grow you can't do it like that anymore. Code will be repeated all over and will be un-maintainable anymore. Not to mention security problems later on, when things change...

Also doing direct SQL calls from an application is without a layer is 'not-that-nice'.....

Anonymous on Mar 1 2006, 07:28

Did you read the code? There is no SQL injection attack possible here. And the idea that because something is procedural it is automatically spaghetti while OOP is not is pretty funny. A template is inherently procedural to me. I have nothing against OOP when used correctly, but I see a lot of things approaching HTML classes where you end up doing `$html->br()` just to spit out a break tag. That makes absolutely no sense to me.

Anonymous on Mar 1 2006, 08:46

I think you guys are missing the point. It isn't about OOP vs. Procedural. I could just as easily have written the controller code as a series of objects. The important point is not the style of the code, the important point is that you don't have a single monolithic controller that becomes your bottleneck and forces all your controller logic into one place because it turns what is supposed to be a discrete and modular architecture into a monolithic one.

Anonymous on Mar 1 2006, 08:54

You could also use OOP without "one monolithic controller".

I've rewritten this application using Ruby on Rails.

DISCLAIMER: I'm fan of rails. Make sure you have some salt to take.

Rails is one of the frameworks which fit more or less in this description:

"Most likely in the form of a big complex MVC framework with plenty of layers that abstracts away your database, your HTML, your Javascript and in the end your application itself. If it is a really good framework it will provide a dozen things you'll never need."

Rails is not big, some things are complex, but simpler than the same thing in PHP. There is only one layer of abstraction for the database (ActiveRecord). It abstracts html to some degree, but you still have "code" templates (like vanilla php), not a separate template language. (like smarty). It abstracts javascript (I did not write a line of JS for this application). I'm not sure what you mean with "abstracts [...] the application itself".

"If it is a really good framework it will provide a dozen things you'll never need."

Ehh, how many functions does PHP have? How many do you use? Providing these is still a good idea, because others need them.

Well, back to the rails version. It is less than 100 lines (excluding CSS). Yours is 400 lines (excluding CSS again). You may formulate a conclusion.

Rails uses the shared-nothing architecture, like PHP, so both are perfectly scalable.

And, in my opinion, the rails version is better designed. You have validation code in the controller. It should really be in the model. Plus, there are 0 lines of sql in my application. This means that you can use any database, mysql, postgres, sqlite, etc. without rewriting your sql.

I agree that your code is not spaghetti. It doesn't mix controller, html and data code (if you don't count the data validation, which is data-code, so that means model)

If I find some time, I will write a more in-depth reply to this. (with code comparison)

Anonymous on Mar 1 2006, 09:42

It's only 400 if you include the Javascript. Otherwise it is 200. Without writing a line of JS, you are not going to get the extra candy effects this app has. The fades, overlays and field manipulation stuff has to come from somewhere.

Anonymous on Mar 1 2006, 10:04

By the way, I agree that data validation should be in the model, but it is called from the controller, and in this simple example all it has is a check to see if all the fields have been filled in. If you need to apply any actual business rules to validate the data, those business rules should of course be in the model.

Anonymous on Mar 1 2006, 10:30

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Maybe I'm out of the loop, but I thought using .inc files was out in the days when ASP just came out. With a .inc file, you can browse to that file from the web and view the contents as text. Let's say you happen to have some password in the code, say for accessing a DB, then that password is out in the open for anyone who cares to find it. All you would have to do is call it 'file.inc.php' and it wouldn't have the inherent problems the .inc file does.

Just a thought.

Anonymous on Mar 1 2006, 10:39

My standard production server Apache config always has:

```
[Files *.inc]
deny from all
[/Files]
```

Replace []'s with angle-brackets. Silly comment system here doesn't let me put them in.

Naming them .php instead and letting people browse them directly can be a much bigger problem since they are now being executed out of context. So you need to either put your include files outside your docroot tree, or you need to block direct access to them with an Apache rule as above.

(and no, this rule is not in place on the talks server where this app is hosted, because the whole point here is to actually show the code ;)

Anonymous on Mar 1 2006, 10:46

Still, I find frameworks much easier to work with (specially with more than one developer) just because they comply to a known structure. In most MVC frameworks the Front Controller or whatever you wanna call it just delegates the request to other more specific controllers so you don't really have one single controller doing all the work. At the individual controller / model level you still have loads of chances to do whatever you want (for example writing your own helper classes and using them however you see fit in your models or controllers) so it is not accurate to say, as it is often said, that frameworks limit flexibility (ok, bad ones might!). They just structure the workflow and allow for flexibility for the things that really need flexibility (like specific business logic operations). If only to my personal experience, this has proven to be true. I also think that PHP is perfectly suitable for good MVC framework design and that it should be encouraged.

Anonymous on Mar 1 2006, 11:10

Scriptaculous is a component of Rails, and includes the javascript effects. No external libraries needed, just a fresh rails installation. The javascript effects are created using ruby code that gets compiled to javascript. There is a feature called rjs (ruby-javascript) templates for these things.

Example:

```
page.visual_effect(:highlight, 'the_id_of_a_div_on_the_page')
```

Gets compiled to something like:

```
new Effect.highlight('the_id_of_a_div_on_the_page');
```

And this javascript code is executed in the browser.

Rails is consistent: all validation in the model:

```
class Product < ActiveRecord::Base
  validates_presence_of :description, :category
end
```

If you try to save a product with an empty category/description, you'll get an error message.

I've studied your php example, and I see that I can strip some more lines off the ruby code: your script just refreshes if the input data is good, my version updated the products table, and highlighted the successfully saved product.

But I really like your approach overall. This mvc 'framework' could even be used for really big applications. You would probably create a few folders (views, controllers, models), but you don't need objects to do mvc.

And I agree that you don't need a template language. PHP was originally intended as a template language, and it is really good for html generation. You have to keep attention not to mix the m, v and c if you are used to spaghetti code though.

Anonymous on Mar 1 2006, 12:21

How about Ruby on Rails instead? I've been studying it for two weeks and love it. Admittedly, I've haven't done PHP, but it is definitely easier to use than J2EE. Worth a look.

Anonymous on Mar 1 2006, 12:40

What about a link?

Anonymous on Mar 1 2006, 12:44

But Rails happen to have some JS niceties build in. I think there is fades and field manipulation but there might be more.

Regarding the actual framework I like your idea that we should write a good base and use that, and there are a lot of frameworks to

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

choose from, but given that you could choose one that you like and use in every project you will probably not have much problem going back to the code a year later.

The naming in your models made me nervous about building larger apps where you need more than one model/table for a view. The can't all have `load_items()` and be included at the same time. Therefore some grouping is needed and I think objects could help a lot. But namespaces might do the trick if someone gets them into PHP...

Anonymous on Mar 1 2006, 12:57

Not necessarily. You can use `.htaccess` or the webserver's ini file to either deny access to `.inc` (and `.db` files, in this case) or have them parsed by the php parser.

But I agree, I also always name my include files `.inc.php`, just to make sure.

Anonymous on Mar 1 2006, 13:29

Hi!

Indeed I overlooked the fact you're using PDO. My bad ;)

I agree that your code isn't really spaghetti code (and we all know how capable you are, let that be clear), and I certainly agree 100% about your comment in regards to extremely bad designed and totally besides the point written frameworks.

So how's Pear doing these days then? I'd say it is a PRIME example of how NOT to design an OO framework, so there you have it. People download pear code, learn from it, and think it's OKAY to have a Date class which in total uses nearly 250kb of code... I rest my case ;)

I think you are doing a lot of damage with this posting. Do you realize what's going to happen if ONE of the 30 programmers in the building I work in find's your post? For years I have struggled convincing fellow programmer's that PHP is NOT a simple scripting language for the gamers and what not, that it actually CAN compete with J2EE and ASP.NET and C# if used where appropriate. To read this from the creator of the language....I'm still shocked

Besides, how this example is supposed to show that using a monolithic controller is bad is beyond me.

Greetings,
Jürgen

Anonymous on Mar 1 2006, 13:51

Sure, JS stuff is built in, but you still need to control it. Whether you do that with Ruby code that generates JS or just with straight JS, you still need to write it. It's not like I wrote reams of code to fade and unfade. It was just a 3-line function for each:

```
function fade(o, dur, fnc) {
  var oAnim = new YAHOO.util.Anim(o, {opacity: {from: 1, to: 0}}, dur);
  if(fnc) oAnim.onComplete.subscribe(fnc);
  oAnim.animate();
}
```

```
function unfade(o, dur, fnc) {
  var oAnim = new YAHOO.util.Anim(o, {opacity: {from: 0, to: 1}}, dur);
  if(fnc) oAnim.onComplete.subscribe(fnc);
  oAnim.animate();
}
```

where I can pass in a function pointer to be called when the animation completes. I don't really see how Rails can abstract that away. How do you tell Rails the starting and ending opacity levels on a fade and the fact that you want a certain function to be called at the end of the fade unless you actually specify this somewhere?

Anonymous on Mar 1 2006, 14:59

This has nothing to do with Pear. You will notice I am not using any Pear components here. And I am not trying to illustrate that a monolithic controller is bad. I am simply stating that as fact and showing how you can build something without one.

I still think you are hung up on OOP vs. Procedural. Both the controller and the DB layer could easily have been OOP and it wouldn't change the point of example nor does it increase readability. I guess your biggest beef is that I used a global for my database connection handle instead of making it a property of a database wrapper class? It's a minor point as far as I am concerned, and yes for namespace-related concerns you may want to wrap it like that, but again, this was not the focus of this article. The focus was the separation and where to put what, not the line-by-line details of the actual implementation. You also wouldn't use sqlite for a huge project, but it is perfectly fine as a way to illustrate the concepts here.

Anonymous on Mar 1 2006, 15:09

Hi

OK, apologies. You seemed to be rather sceptical about OOP frameworks. I'd go along with that for the most part. Just because it's object-oriented doesn't automatically make it good. It's hard to find well-written examples in php.

I'd agree with your point about templates (and html classes which emit br tags...) although I'd definitely have a whole pile of objects doing all the validating, data gathering, business logic etc prior to throwing a few vars at some simple php presentation code in the template.

I find I'm not sure exactly what you mean by "monolithic" controllers though. To an OOP bigot like myself, there's a choice between

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

FrontController (single point of entry) or PageController (multiple points of entry) but I can't see any compelling reason to choose one over the other. FrontControllers can hand over to any number of objects to carry out the application controller logic for a given request (that's their only unique role otherwise carried out by apache) and PageControllers can extend a superclass which provides a single location for common processing tasks. Six of one, half a dozen of the other. I tend to prefer PageControllers but I think you can write a flexible FrontController design - and you can have more than one FC if need be.

Anonymous on Mar 1 2006, 15:49

My problem with monolithic controllers is that I don't like centralized presentation-level logic. Views and their associated controller logic should be as discrete as possible. If you start lumping all your controller logic into one big pile it becomes very hard to swap out individual views or change the flow slightly in just one part of the application without affecting other parts.

So the gist here was small discrete controllers, clean HTML templates where the HTML looks like HTML and a clean and separate data model layer. The point wasn't so much to say all frameworks out there are bad or that OOP is bad, more to say that it isn't hard to build your own base that does just enough for your particular problem and uses whatever style of programming you are comfortable with instead of having to try to fit your problem into someone else's idea of how you should structure your code.

I was also hoping people would pick up on how the Javascript code manages to connect events to the various form elements without having all sorts of onclick/onmouseover stuff in the templates along with the various performance hints related to opcode caching and storing lists in shared memory.

Anonymous on Mar 1 2006, 16:12

I actually thought this was very refreshing.

Sadly, I feel confident in saying that the PHP frameworks that are here today, probably won't be in 2 years. That's what I'm getting burnt out on. One thing that I'd love to be able to do is come back to an application in 6 months, and know what the hell is going on. I can say for sure that I wouldn't want to read about how the "framework" works. Ugh.

I've been trying out PHP frameworks for at least 3 years and now... I love to see all of the oop/framework purists squirm! PHP does a great job when used simply. Nice work.

Anonymous on Mar 1 2006, 18:58

I learned my lessons about monolithic controllers the hard way by having to maintain my mistakes. It's not even Legacy code and it's a pain in my a\$\$.

Hope you post more articles, more often. The PHP community could use someone pushing simple, small and fast.

Anonymous on Mar 1 2006, 21:40

I am not sure where you needed a callback, but this is my "javascript" code:

```
visual_effect :fade, 'modified', :duration => 2
```

Anonymous on Mar 2 2006, 00:39

Perhaps this comes down to a question of control. In your case, where is this Modified element? Is it relatively or absolutely positioned? And the callback is there to reload the form and possibly reset it if that was asked for. I don't see how that one line of yours replaces:

```
div = document.createElement("div");
div.className='status';
div.innerHTML = resp['status'];
pos = YAHOO.util.Dom.getXY(resp['elem']);
div.style.visibility = 'hidden'
document.body.appendChild(div);
YAHOO.util.Dom.setXY(div,[pos[0],pos[1]+40]);
div.style.visibility = 'visible'
fnc = function() {
  if(resp['reset']) { document.forms[resp['formName']].reset(); }
  window.location.reload(false);
}
fade(div,2,fnc);
```

unless something is reading your mind somewhere.

Anonymous on Mar 2 2006, 00:51

The code would look something like this (still Ruby on Rails):

```
{ :action => "add" },
:update => "modified",
:complete => visual_effect( :fade, 'modified', :duration => 2)
)%>
```

That's all the JS/AJAX code necessary to update a given element and to add a visual effect such as a fade-in or an animation of some sort. Much easier to read IMHO.

Anonymous on Mar 2 2006, 05:53

I see no mention of a div there, so the two bits of code obviously don't do the same thing. Mine creates a div, fills it with something,

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

positions it absolutely, appends it to the document, then fades it and after it fades the page is reloaded which clears out the element and the form fields and updates the form.

Anonymous on Mar 2 2006, 06:35

I do agree with you that writing Javascript in Ruby like this is much easier to read. Javascript is an annoying language to use. But I lump it into the same category as HTML. Writing HTML is annoying too, but trying to abstract away the annoyance has never worked for me. HTML should look like HTML and Javascript should look like Javascript. For very expressive things like HTML and Javascript you end up inventing an entirely new language to try to represent everything that can be expressed in those.

Anonymous on Mar 2 2006, 06:40

What that code does is makes an ajax call to an action called "add". That action returns a html-snippet, which the element with the id "modified" is updated to contain (replacing any previous content inside it). Then the same element is faded in.

Another way to do it would be to just add a html snippet to the bottom or top of an element container (instead of replacing the contents entirely), simply by adding `:position => "bottom"` to the code.

Except for some insignificant differences, the code does what your code does (in about 200 lines less).

Anonymous on Mar 2 2006, 06:48

Well, you could argue that it replaces the 13 lines of Javascript, but it isn't 200. The 200 lines of Javascript does all sorts of other stuff. And I would argue that the differences aren't insignificant and in this case probably adds another round trip?

Anonymous on Mar 2 2006, 06:57

```
{ :action => "add" },
  :update => "modified",
  :complete => visual_effect( :fade, 'modified', :duration => 2)
) %>
```

Rasmus,

The `:update => "modified"` portion is basically telling the page to replace the contents of any element on the page that has an id attribute of "modified" and replace its contents with what is returned from the AJAX call. There are other options you can provide that would allow you to add the new stuff either at the top, bottom, before or after the specified element.

I might suggest reading the documentation[1] before arguing that it can't be done.

[1] <http://api.rubyonrails.org/classes/ActionView/Helpers/JavaScriptHelper.html>

Anonymous on Mar 2 2006, 07:05

I didn't say it couldn't be done. I said the example shown doesn't do the same thing as the code it supposedly replaces. I have no doubt that you can do the same thing, but the 1 or 2 liner shown doesn't give me the same level of control.

Anonymous on Mar 2 2006, 07:15

No, that's not correct. I know PHP very well, and I know Ruby on Rails quite well. For you to have the complete perspective, you'd have to learn Rails as well.

Anonymous on Mar 2 2006, 07:52

This particular point has little to do with PHP. It's a Javascript thing. My additional roundtrip comment had to do with the fact that you only did the Modified message there, but not any of the other actions, so you'd have to make another trip, or do something different from that one-liner.

But this is getting a bit semantic. If you prefer writing Javascript in Ruby and that works for you, great. I'll stick to writing my Javascript in javascript.

Anonymous on Mar 2 2006, 07:59

Great, now I know how to do a fade!! Thanks guys!

Anonymous on Mar 2 2006, 11:16

How did I know this would turn into an OOP vs. Procedural war? Okay, maybe war is a strong word, but the new disclaimer at the top from Rasmus speaks volumes.

Well said Buddha. If the Java vs. PHP argument means you have to turn your code into something that looks like Java, then you might as well use Java.

Anonymous on Mar 2 2006, 16:24

Your article just confirmed the impression that I had when trying to make one of those monolithic controller. I was jailed in that controller and could not even do what I wanted to do.

Now I changed my way of coding, putting the controller away and now I'm free to do more in less time. Maybe I'm not a super MVC guru but I'm free.

(If you find my comment quite unreadable, it's normal: I'm from Quebec)

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Anonymous on Mar 2 2006, 19:33

It is quite common for knowledgeable web admins to have .inc files parsed by the php interpreter (unless they use server-side-includes on the same webserver as php).

It is true that if every php coder had used `included_file.inc.php` from the beginning, this would not ever have been a problem...

Anonymous on Mar 3 2006, 02:42

"It is downright scary how much we think alike. Even your code looks so much like mine you'd think I stole everything I do from you--which in a way is partially true"

I think I can fully subscribe to the above post.
Hire me! ;)

I know OOP and MVC purists will frown at the code, but I really have a hard time 'getting' the benefit of turning all (and I mean all) functions into objects and adding layer upon layer of abstraction for the sake of 'architectural style'.

In my own experience, when a new coder is brought on the project 2 years after the initial one left, most of the time it will in fact be easier for him to grasp what a single function does and what it should not do than fully understand the delicacy of object manipulation - especially if the object coded by guy nr. 1 is poorly documented and makes a lot of assumptions about its usage patterns.

As for the framework craze, they generally have a very steep learning curve. The main benefit arises from the fact that you can expect the next-programmer-in-the-row to know the chosen framework and understand it better than the code you have developed in house. Unfortunately, they come and go so quickly that it is more than likely that this will not happen.

My 2c.
gaetano

Anonymous on Mar 3 2006, 02:58

You still need to be careful about that. Executing a .inc file out of context can be a security problem. You are better off placing them outside your docroot or turning off direct access to them in your Apache config.

Anonymous on Mar 3 2006, 07:49

I find your article somewhat refreshing in the fact that if you want to separate your code along the "MVC" lines you can do it simply without using complicated frameworks. And if you don't want to separate it out (leaving your controller code in the view itself) then that is fine too.

That is what makes PHP a joy to code in. You are not hog-tied into doing things a certain way. If you want to make your application complicated by developing your own or using an existing framework, go ahead. More power to you. But I like to keep things as simple as possible. And I totally agree with Rasmus. HTML should look like HTML, Javascript like Javascript.

All I know is that, when I go back to work on code I wrote years ago and all I did was page-based development with the controller at the top of each view, it is fair easier to debug and maintain than the complicated mess you can get into by abstracting your application and splitting up all the elements that make up a request across a billion different files.

In my book, simplicity always wins out versus complicated.

Anonymous on Mar 3 2006, 08:17

"Monolithic controller" sounds as if you were doing something wrong. As I said, controllers should be focalized and simple, and you achieve that by separating responsibilities with the correct use of objects.

True, the learning curve to most of frameworks may be steep, but once you really understand their structure (which is similar to most) it gets much easier to take someone else's work, just because you know where to find things. You know what to expect from the framework. It is not about piling up layer after layer for the sake of it. It is about a scalable structure. When everybody knows the structure it can only be a good thing, and that's what patterns are all about.

Anonymous on Mar 3 2006, 11:24

Sorry to comment again, but I was wondering if along with your example of simplifying MVC, you could throw at us any certain type of directory structure that you use. I see how structured the directories have to be with Rails, and other frameworks, but using your approach, do you have any certain directory structure for your files that you like to use?

Again, nice article, great way to break MVC down simply without all the objects.

Anonymous on Mar 3 2006, 11:40

I think that's really personal preference. I like to have any code that has HTML and CSS in it in one place and the rest elsewhere. In my example I did that by moving the model code into a `model/` directory. In some of my full applications I put all the backend business logic (model) code outside of the docroot on my `include_path`.

Anonymous on Mar 3 2006, 12:30

Rasmus, I was hoping you could answer a question. In your article you say "I tend to use separate include files for each table in my database." In your example with 1 database that works out very well. But what if you have to be able to manipulate some more tables. Say for example an webshoplike application. After posting a form, several tables like `items_in_cart`, `items_in_stock`, `orders`, etc etc must be updated. Would you include a separate `db.inc` for each of those? Is there a point when you would take a different approach? What is the advantage of using separate files for each table, compared to one common database class with methods to select, add, delete etc?

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Ok, this turned out to be more than one question, but I hope you get my point and shed some light on this.
Anonymous on Mar 4 2006, 00:30

Well, I should say a separate file for each set of related tables instead I guess. If the same set of tables is always used together, lump the calls to manipulate them into the same file, but separate things that are logically separate. From a performance perspective you are better off with less files that are larger than more files that are smaller.
Anonymous on Mar 4 2006, 06:42

I've nearly finished the explanation of the Ruby code. Because I wanted to get Railsish coding style, I haven't translated your code line-by-line. I've looked at the example application, and used that as a goal, so there might be things I overlooked. It is very possible that my application doesn't do the same as yours, but from the user's perspective, they are functionally equivalent. When the user enters incorrect data, the missing fields will be highlighted. If the data is good (all the fields are filled in), the user will see the updated list, and a fading "modified" text. I've positioned this text relatively, but it could easily be positioned absolutely with CSS.
Anonymous on Mar 4 2006, 10:26

Port to Rails, with code + explanation (no ruby course though ;-)):

<http://www.web-site-build.com/no-framework-application-on-rails/>
Anonymous on Mar 4 2006, 12:37

"From a performance perspective you are better off with less files that are larger than more files that are smaller," on a shared server, all file operation seem costly.

Trying to keep file operations small and few has lead me to build the PHP files with only data and functions needed (I use a perl script to do this). This results in PHP files optimized on a per request basis but with the drawback of lots of code duplication on the production server. There other benefits when it comes testing, maintenance and code reuse. It has developed into metaprogramming.
Anonymous on Mar 4 2006, 13:57

you should try phpontrax.com if you like RoR's but want to use php. It has all the scriptaculus stuff built in same function calls as RoR. Syntax is almost identical to RoR.
Anonymous on Mar 5 2006, 07:08

After reading chapter 14 of the PHP5 Power Programming book, in particular pages 467 and 470, I got the impression that an opcode cache like APC would "tuck away" the `require_once` and `include_once`. You seem to say here that `require_once` are slow under an opcode cache, hence contradicting what I had understood. Can you please clarify this point?
Anonymous on Mar 5 2006, 12:10

The opcodes for the included file will be "tucked away" in shared memory. But we still need to identify which opcode array to pull out of shared memory on an include. When running under an opcode cache `include_once` and `require_once` do an extra `open()` syscall compared to their `include/require` counterparts. Of course they end up not actually reading from the file, but because the functions are geared towards a non-opcode cache environment there is a bit of a mismatch. I will eventually come up with a workaround for this. Maybe by adding an extra hook, but for now you will take a slight hit if you use `include_once` instead of `include` everywhere.

Don't get too depressed though, it's not a big hit, but if you are trying to squeeze every last bit of performance out of a system, this is one of the places where you can save a syscall. Have a read through this for some more APC optimization hints:

<http://news.php.net/php.general/231433>
Anonymous on Mar 5 2006, 12:25

What do you think of the Zend Framework that was just released?
Anonymous on Mar 6 2006, 07:28

Their point is it does - they chose to update an existing div, but you could also create divs.
Anonymous on Mar 6 2006, 09:31

Is there a zip file for the code in the example - didnt see one?
Anonymous on Mar 7 2006, 10:43

There is already a PHP implementation for an MVC Framework that works like RoR and already has the basic functionalities that RoR does like, ActiveRecord, Ajax, Html objects, JS Effects.

There is a sample blog application along with it, to let you experiment with the feature. Though this is still in its pre-alpha release, but early evaluation will determine if it looks promising or not.

You might be interested in taking a look at `phptarsier` (<http://www.sourceforge.net/projects/phptarsier/>).
Anonymous on Mar 8 2006, 00:44

If you're looking for RoR inspired frameworks you might want to check out <http://www.codeigniter.com> as well.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

- very easy installation (No PEAR, just unzip, update config and ready to get coding)
 - source code easy to understand
 - clear and concise documentation
- Anonymous on Mar 8 2006, 03:12

Rasmus, have you had a chance to play with the Zend Framework (0.1.2)?

I'm very curious to know whether you will stick with your current approach to building php apps (which I very much like by the way), or if you'll adopt the Zend Framework once it becomes more stable.

Anonymous on Mar 9 2006, 07:56

Looks like all member functions of class item could be static. Any particular reason why they are not?

Anonymous on Mar 10 2006, 23:38

They can't actually be static the way they are written since they require an instance of the db class which carries the database handle. The idea being that each instance would be a separate connection.

Anonymous on Mar 10 2006, 23:49

I thought I made my feelings pretty clear on FrontControllers ;))

Anonymous on Mar 10 2006, 23:50

isn't keeping multiple connections expensive? can all members of the db class be static, too?

Anonymous on Mar 11 2006, 00:17

Sure it is expensive, but if you need multiple connections, like when you have different physical databases to talk to, then there is no way around that.

You can simply make \$dbh a protected static to force all instances to use the same database handle and then change all \$this->dbh to self::\$dbh throughout. No real point in making the methods static since you are still likely to carry some other info with each instance, but with a static database handle you get the single connection. I actually have it this way in CVS, but it isn't showing up here because of the annoying way CVS handles symlinks. I'll fix it.

Anonymous on Mar 11 2006, 00:33

You did :))

Anonymous on Mar 14 2006, 06:36

Lee - Here's the (Rails-like) directory structure I use for framework-less PHP:
<http://jonaquino.blogspot.com/2006/03/what-makes-rails-great-its-directory.html>

Anonymous on Mar 16 2006, 09:40

I think that the discussion brought forth by the article is depressing. There are so many people that claim to know the "right way" to develop in PHP, and it's scary to see how people defend those opinions using "rational" arguments

ra tion al (adj.) Devoid of all delusions save those of observation, experience and reflection.

The saddest part is that 90% of the comments are attacking the implementation rather than the concept. Who cares about the database? He could have used a flat file for all of the difference it would make to the concept. And if you can't realise that with all of your talk of abstraction being a key factor in a Framework, then you really are clueless.

At the start of the Article, Rasmus outlines his goals. The first question you should be asking as a supposedly intelligent person is "Did he reach his goals with his design?" I think he did.

The next question you should be asking yourself as a supposedly intelligent person is "Does he forward anything in this article that I can put in my own knowledge base and perhaps even apply to something I am doing?" If nothing else it got me to look at the Yahoo API and so the article was worth it.

To sum up, get over yourselves. Look at things with an open mind unless you feel you are the best programmer you can be. If you do feel you are the best you can be, send me your resume so that I can be sure NOT to hire you.

Leave debates over the mean of "is" to the philosophers and get real.

Matthew.

Anonymous on Apr 14 2006, 11:28

Mmmm, I wandered over here after a thread on the Zend Framework list.

I have to say a lot of people have seemed to have missed the whole point of this. Its simply an example of how you could do things. But you are free to choose, you are after a php coder not a mindless sheep.

Personally I have some problems with certain frameworks, yes I do understand the need for them to be there. I think that a lot is to be said for coding standards and just because you don't use a frame work does not mean you can't avail of the reusability of designing with OOP too.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

All in all, I very rarely comment on PHP stuff cause there is someone out there whom always knows more, but seriously this article gives a very good example of how you can achieve your end goal relatively easily, yes you may very be able to pick holes, but that could apply to the framework your using too. In short, use what you think works for you and your client. But do document everything :) and if you can use a coding standard, its so worth it.

Anonymous on Apr 16 2006, 16:04

Sometimes I use defined like in C headers to avoid the use of include_once in situations like this.

```
If the file filex.php may be included more than once and there's seems to be no way to rewrite the code to avoid include_once then I use define('INC_FILE_X',true) at the top of filex.php and instead of include_once if(!defined('INC_FILE_X')) { include 'filex.php'; }
```

Quicker than an open() syscall right?

Anonymous on May 11 2006, 03:33

Hello,

Great article!

Thanks

Anonymous on Jul 24 2006, 05:25

I found this site and was excited to see how to go about doing MVC w/o a framework, because of all the arguments that are made in the intro to this article. I'm also very new to PHP, so the more examples I can find on how to do things the better.

To my disappointment, the code with this article doesn't appear to be available. If the code is supposed to be in the IFRAMES embedded in the article, then that is a disappointment too, since some of them are being rendered (and they appear to be broken at that), some are code, but its all presented ugly and unformatted (which I could deal with, but only part of the code is here).

Anyone else have these issues? Can the site be fixed? Could the code be available as a separate download?

You have me interested, but I still don't have enough info to see how much effort is involved vs. learning something like Zend.

Anonymous on Aug 7 2006, 06:55

That should be fixed now. Server misconfiguration. You might have to clear your cache though to get the iframes to show up correctly.

Anonymous on Aug 7 2006, 08:42

nice article

Anonymous on Aug 13 2006, 12:22

Rasmus,

I read this article back when it first came out and thought it was awesome. Because I had spent many months on researching "frameworks" that were mostly OO and a lot more complicated. After coming to the realization that, the apps that I wrote 5 years ago still work fine, and they don't use any framework, I wanted to forget about the whole framework crap and get back to improving my current apps in place.

Well shortly thereafter, the Zend Framework was released and I decided to try and learn that, since all I was hearing was it was going to be the way PHP apps are built in the future. Well, after many months of playing with it, and after deadlines for releases are passed without any releases, I have just about giving up on it totally.

Then, when I was cleaning out my bookmarks one day, I found the bookmark for this article. Very refreshing after all the months of time I wasted on trying to learn a framework, that I have decided to go back to what you originally suggested as an approach to MVC in this article. Heck, I don't even necessarily need MVC...I mean I work alone, don't have a team or anything, so I can pretty much do what I want. And what you suggest is so much easier and what i'm use to, than any of those "frameworks".

Coming from one person who has invested the time in learning numerous frameworks...I'm back to the real world, getting real work done! Thankfully.

One question I do have for you is this. When implementing authentication and authorization, how would you go about that in your suggested MVC approach?

Once a user is logged in, would you have a function run in each controller checking whether they are logged in, and then another function for authorization to check whether they are suppose to use that particular section of the app? Just curious. Thanks!

Anonymous on Aug 16 2006, 13:09

Right on. Straight line cast the problem without a bunch of crap.

Anonymous on Jan 2 2007, 10:50

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

The greatness of PHP is the freedom to choose between simplicity and complexity. That's what made PHP the king of the web, and that's what this article is all about.

Anonymous on Apr 4 2007, 02:02

I am open to Framework and I agree with your article. For some case Framework increase complexity and make team confused especially website designer and when new developer join with running project.

PHP is the fastest development language I ever use, the simplicity of PHP make the programming could follow the logic easily. I always use the way you describe to specific environment, I don't want to add extra effort when I could develop an application with all built-in functions provided by PHP.

Anonymous on Apr 19 2007, 21:00

I must agree with you Federico, Objects are a problem in programming/scripting these days. People try to force usage of it. That's what's wrong. It wouldn't be the first time that I was called a newb cause I refuse to use OO in certain situations (Simple things that don't require OO at all). Give people the choice of programming style. It's not cause you like Ruby on Rails that we need to like it. I hate the whole Rails thing from line 1. I've tried it and I got sick of it after less then a hour. rm -r was the best project solution for Rails in my case. If you like rails use it but don't try to force it on to other people.

Also try to benchmark code that uses a lot of objects and the same thing in procedural code. You'll be amazed in how many cases the procedural code will win cause no useless definitions were made.

Side note: Before you also start saying things like "You just don't know how to use OO", I do use OO; I just don't use it when its not needed. OO is great for handling connections.

Anonymous on May 4 2007, 09:20

Althought I agree with Rasmus' point of view, usual tasks lead me to write my own framework. At least I know it very well and I can change whatever I want !

Anonymous on Jul 29 2007, 01:24

When you get down to brass tacks it is all about using the right tool for the job, sometimes that tool is OO and sometimes it is not. The great thing about PHP is that you can use it to create the tools that you need when you need them.

The single most important point here is that you should make your code modular. In the simplest state this is a Model module, Controller module, and a View Module. My customer Model is different from my Shopping cart Model, so why would I want them both handled by a Master controller (regardless of OO or procedural)? For some apps this would be OK, but if you want scalable you must plan on implementing modular code. (BTW, RoR is probably not the most scalable -- handling thousands and thousands of requests per minute in a mission critical web application -- framework that I have ever seen but it sure works well for certain sites.)

Personally I find that too many of the frameworks add too much complication to an enterprise web app design. Within our team we have specified standards and that is the framework from which the team operates. Other than that they use MVC pretty much as Rasmus has described here. Sometimes the Controllers are a little too close to the views, but they are easy to maintain. You can completely re-design a View and Controller for a Model and pop it into place without much hassle...usually just by replacing 2 or 3 files. In the mega-Controller, unless you organized it really well, this would not be the case.

Another thing for me on the large Controller; if all of your applications users are hitting it all at nearly the same time you will suffer performance degradation, especially in the very large scale applications where 1000's of connections each minute are possible.

Ah well, just another rant for a Friday afternoon. As Einstein once said there is beauty and elegance in simplicity. Thanks Rasmus.

Anonymous on Sep 7 2007, 14:06

Rasmus, I am in agreement with the initial intent of this article. If I hadn't taken the time to read all of the negative comments here I probably would never have bothered to take the time to voice my opinions here. My only hope is that anyone reading this will realize that some of us older guys may just have a better grasp of the realities of programming than some might think. Separate the logic but don't make things any more complex than they need to be.

I stumbled across this article while doing additional research on MVC and OOP while developing a plan of action for for a new CMS project which will supersede my current CMS, Etomite. I have to agree with several respondents in that many of the complaints here are not addressed to the original intent of the article.

I use PHP every day. Some days I need to use Javascript. These two languages are a drop in the bucket compared to the number of languages I have used over the years. I am currently debating the use of YUI as in integral part of the new code base for the new project. With all of the research I have done, and testing of software packages, I keep coming to the same conclusions. Some folks just like making their lives more complicated than they need to be. Some explanation is in order.

This whole idea of Web 2.0 (AJAX) is a prime example. Is it really anything new? Not really! It's simply finding new things to do with previously existing technologies.

Do these new features make developers lives easier? Not really, because you have to learn the new code library. Yet more syntax and semantics.

Same goes for frameworks that rely on too much OOP. Learning the framework, or the multitude of classes, can take longer, and cause more development problems, than simply writing good code. Sure, it all works well and looks real fancy once you know your way around and get all the bugs worked out. But it may have been a whole lot easier to just write good code using what you already know rather than thinking that new cryptic pseudo-languages are the answer to everyones problems.

Performance is yet another issue that needs to be considered more than it has been in recent months. All these supposedly new technologies, or the over-use of old technologies, has brought parts of the internet to its knees. And don't even get me going on the

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

lack of compliant code being wretched out due to the use of these splendid ideas that people have. And, yes, even too much OOP will slow down your application performance, as well as development time line.

Sometimes it's better to use the KISS concept. Granted, there are times when a framework might make sense - but make sure you really need to go all out. Simply knowing something doesn't make it the right thing to do for every task.

Again, great article, Rasmus...
Anonymous on Sep 28 2007, 11:49

I completely agree with Rasmus on not just on the single monolithic controller point but also the no brainer ideology of 'everything should be an object'.

I've been into web development for 7 years now, and currently managing quite large teams of developers and I have learn't the harder way that usage of OOPs where it is not needed has such an adverse effect on the developer productivity & code maintainability that it brings down PHP to the level of JAVA & .NET

One indicator that you are misusing OOP is when you write a class without considering a design pattern for it. Like, when you don't need an observer or factory or singleton etc. for a task, let it be procedural. Or, you will end up with code like `$html->br()` as Rasmus rightly mentioned.

Maximum conceptual integrity, maintainability, re-usability & simplicity can be achieved when you write classes with perception of writing re-usable tools for usage within your standards compliant procedural code.

Consider this procedural controller (generated by my MVC code generator) from the maintainability point of view:

```
include('init.inc');
include(LIBRARY . 'records.inc');
include(LIBRARY . 'authentication.inc');
include(LIBRARY . 'forms.inc');
include(LIBRARY . 'class.genericformvalidator.inc');
include('models/register.inc');

$action = $_REQUEST['action'];

InitModel();

switch($action)
{
case 'register':
    $RegistrationInfo = ProcessRegistration();
    if(count($RegistrationInfo) > 0)
    {
        if(ProcessLogin($RegistrationInfo['username'], $RegistrationInfo['password']))
        {
            header('Location: memberships.php');
        }
    }
    break;

case 'failed':
    DisplayErrors($_REQUEST['errors']);
    include('views/register.phtml');
    break;

default:
    include('views/register.phtml');
}
```

Note the 'class.genericformvalidator.inc' in the includes, it is a class I wrote for form validation and is used for a form in register.phtml. Now why did I write only that in OO? because I wanted there to be one central validation engine which can be extended by custom validation classes. OOP is a great feature unless its used just for the heck of it.

Rasmus, this is an excellent article and I enjoyed reading it. Just ignore the object orient idiots, they will eventually learn the harder way that you were right.
Anonymous on Oct 15 2007, 04:37

I'm not convinced any of the php frameworks make things that much easier.

At first glance, it seems so. But instead of understanding code in one place, you have to go looking at code in a zillion different files all over the place. Presentation is in one file, db connection in another, validation in another, javascript somewhere else, etc. etc. You end up including a bunch of shit you'll never need....bloat, bloat, bloat.

How is this easier? You have to cd back four directory structures just to see where the validation is taking place? Then the frameworks always have to be extended and going through someone else's code and customizing it is always a pain in the ass.

I much prefer a lightweight framework as Rasmus is suggesting and separate a few things where necessary. But these frameworks are waaaay out of hand. Have you see the websites using some of them? Sloooooow as hell.

`$html->br()` is classic crap you have to do in many of these frameworks.
Anonymous on Oct 16 2007, 12:19

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

I really agree with Rasmus, it's a proven and brilliant idea, I have already proven it to myself. I have been using PHP to build website for 4 years using simple PHP programming, and it works very well. Last year I have spent most of my time learning PHP frameworks and looking which is best but I ended up quitting, I never have use any of those in building a website, It's just too complicated and hard to maintain.

I prefer to make it simple and less code as much as possible.

Clean, secure and fast, that's it!

Great article Rasmus!
Anonymous on Jan 30 2008, 22:32

Bravo! Some sane reasoning for all this recent PHP framework madness. PHP elegance comes from it's simplicity and the ability to be used flexibly. And this framework that Rasmus illustrates, clearly illustrates the only real framework you need. It's simple PHP at its core, just separated into the basics of MVC.

Any PHP developers that have been doing PHP long enough has developed their own frameworks, of course they are not as abstracted or generalized as the big one (PhpCake, Symfony, Seagull), but they get the job done.

Agreeing with most of the other posters, frameworks abstract the development process so much, that you'll end spending more time configuring them, looking for file(s) and overall managing the development process instead of actually developing. The time savings in my opinion are negligible and the developer time lost in learning, training or retaining the framework structure is much greater than stock PHP.

When it's all said and done the bulk of your coding is in the logic, the controller piece, no framework in the world simplifies that. The only "big" framework I would consider is CodeIgniter, it's the one the most closely resembles a grown up version from what Rasmus illustrates here.

Programming is hard enough why add more abstraction?
Anonymous on Feb 7 2008, 08:04

What about of Smarty in the view?, I think it will cause an extra overhead on the server...

nice post.
Anonymous on Mar 4 2008, 19:58

Thanks for the comment Rex S. I was about to throw myself into Framework Hell but was doubtful about how much use they would be long term. I am currently working on a very tightly frameworked project in .NET and it takes a very long time to make any changes and it is hard to see the logic in one place. The original code generator is no longer used and structures are now coded by hand - it takes forever and is no fun whatsoever.

For a comparison my previous project was in ASP and the majority of the logic was held in single ASP pages, maintenance was a breeze and performance was good (the projects were functionally very similar).

As a comparison I would say a similar change would take 1 hour on the non framework system and about 1 day on the frameworked system. OK the frameworked system was allowed to get out of hand but when you get a number of people doing their own thing anything can happen.

I still think a framework/code generation can be a good thing but you have to watch out for the point where it becomes a hindrance – and it this point you are probably too reliant on it to back out or abandon it.

I think Rasmus has demonstrated a viable approach to flexible development philosophy and on a more careful read though of his example it looks even better. I will be trying out his methods and will try to sneak some code generation and framework advantages alongside it. Hey – I want my cake and eat it. The jury is still out for me. Great article thanks.
Anonymous on Mar 6 2008, 02:19

I can confirm the ideas of this article and the comments of those who agree with the objects of the article. I am living proof.

I am a PHP noob, an OOP noob and an MVC noob. You may already be thinking about dismissing my opinion before I can explain my position, pls bear with me.

I have been building websites (small ones), over the past four years using Dreamweaver, for both static and dynamic projects, (this is an instance where size DOESN'T matter, trust me). It was only recently when I have been involved in the creation of one of my own projects, that I decided I would have to, do things as I did when I first became interested in computers over twenty five years ago, that is, get my hands dirty in some heavy code.

I needed a site that would through up articles and video/audio that was related to the article. I needed to create a tutorial website, that makes heavy use of tracking users actions, article use over time etc. I first looked at the two major open source frameworks, but I soon ran into obstacles when I wanted to do something specific. I spent over three months becoming familiar with both of them before realising a simple truth.

Then I spent another month researching PHP frameworks, having been sold the marketing mantra of, access to libraries, cleaner and better, more organised code. Again after trying to wrap my head around MVC and OOP, I realised that same simple truth. What is this truth I speak of?

I spent so much time learning these CMSes and frameworks, that I believe I could have completed the the whole project within the same time period, using PHP5 classes and straight HTML/CSS.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

As for cleaner organised code that you can come back to in a year? I don't understand the argument. If you split your code down into manageable block, as in the example of this article, managability is not an issue.

As for code re-use? again, I don't understand. Do people not build up a library of common classes that are well commented? can you simply not include ir call these classes in your future projects or the same project?

Enough of this nonsense. It seems people against this Rasmus' comments are simply wishing to defend their particular way of working.

I realise I am a guy developing on his own and the needs of a team might be different, but I doubt it.

OK, I'm off to do what I should have done three months ago. Thanks Rasmus.

PS I found CodeIgniter the lightest framework out there and it doesn't impose too many restrictions.
Anonymous on Apr 14 2008, 09:51

Personally, I don't like frameworks. I tend to spend more time trying to figure out the framework than getting anything useful done. I don't see a problem mixing php & html. Web pages are not that complex. Get some data and display it. Validate some input and save it. Abstracting everything out becomes a total nightmare when you try to debug it. Not only that, if you ever take over a project that uses some framework, it will take forever to figure out what the hell is going on. With php,sql,html all in one page, it is obvious and simple and for me, waay easier to maintain. I don't have to go looking for the database class here, the object class over there, blah, blah , blah.

Plus those frameworks are slow as a dog. No thanks. I stick to regular old php and a few common libraries.
Anonymous on May 8 2008, 19:55

I've been studying / comparing / pondering / banging my head on the wall as to which one of the frameworks to go with. I've heard good things about a handful of them, but none are to my liking. They all make simple stuff kind of complex. On the other hand, complex stuff is easier to manage and handle, but I think most of the code I write is very simple. If I had to choose, I'd choose Codeigniter, because it doesn't force you to do things "their way", and it's lightweight.

Very nice article, refreshing, interesting and informative! Reminded me that I don't HAVE to choose a framework to be effective.
Anonymous on May 20 2008, 04:18

Hi Rasmus,

I don't know if you'll write the next edition of "Programming PHP" but personally I would like you to write it, maybe you can extend this no-framework in the book. Thanks for this great article!
Anonymous on May 28 2008, 09:17

I think we can all agree that separating SQL and XHTML out of the PHP is important, and that adding too many object layers not only slows things down but adds far more complexity into getting things done. ZF solves the first problem with the divisions, but then disappoints many with its complexity and speed issues. So everyone after awhile starts to look somewhere else. But the problem with that is that there are so many other frameworks out there now, many of them lightweight and popular, and no real standard, and so you have to keep relearning things. This has got to gestate a little in the marketplace or it's going to be a real mess.
Anonymous on Jun 3 2008, 10:07

Bravo Rasmus. I think some day everything will be object oriented and we'll have to start looking for the 1ghz of ram hosting solutions because of the Ignorant RoR approach to making everything, large or small, completely overdone for the fear of obfuscating into spaghetti. I am very much a fan of OOP, and use it even when unnecessary sometimes because I love the high level of it's product, and it's unrestrictive low-level access from within your application. Even on my computer, I have 10 folders for everything I do because I can't save a honeymoon picture of me and my wife kissing in a folder called pictures, I have to put it in Trips / Jamaica / Pictures / kissing / picture.jpg. All that said, and I still think that over-organization of code, and too strict of conventions can be inefficient, and restrictive.

Thanks for the great article Rasmus.
Anonymous on Jun 5 2008, 22:01

I agree strongly with Rasmus' approach. People say all the time, "RoR doesn't take away from flexibility, or strip you of your freedom. It just gives you a set of guidelines that you can break if you want later." NOT REALLY. What it gives you is a combination of the almost slowest compiling programming language in the world OBFUSCATED, and slowed down even more. If you develop an application in PHP optimized the best way possible with a ROLL-YOUR-OWN simple framework, and run it VS an RoR application that has the same specs, and is also optimized the best way possible, the PHP version will outperform the RoR version by almost 3 TIMES!

Anonymous on Jun 11 2008, 18:21

I have found the above example and everyone's responses to it enlightening to say the least. This re-enforces what I suspected in the first place- that the desired performance and functionality can often be achieved without the need for additional layers of abstraction. Thank you for the article Rasmus.

Anonymous on Jul 18 2008, 07:45

[Language wars]
PHP vs ROR!!!

Anonymous on Sep 3 2008, 01:15

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Michael, that is exactly what happened to me. I was directed to rebuild a rails app in php and when we profiled it, we found the php version consumed over 3x less cpu per instance (about 4ms vs 14ms). Also without the configuration nightmare that is apache proxy -> mongrels (sucking up RAM, memleaking), our deployment process is so much easier too. Also what was an entire rails app, spread across tons of files in several directories at various levels of depth from the project root, became 1 controller, 2 models, and a few view file templates in my larger php project (custom MVC framework). It means less code to maintain, less places to hunt for bugs, less memory consumed, less cpu. Nobody can convince me that php isn't the better if not best choice for web development. Maybe some python zealots ;)

Anonymous on Sep 5 2008, 02:58

I have experienced the issue of learning and relearning as you move back and forth among frameworks. If I used this example as a literal guide to build a website, would you still do everything the same? Would you use PECL, Yahoo libraries, JSON? Two and half years since you wrote the article, is there anything you would update?

Anonymous on Oct 18 2008, 12:36

Great article.

I really enjoyed reading.

I tried python, I tried ruby...

But I always got back to PHP.

I prefer clean, pure PHP code

using objects only where I really benefit from them.

I call my models components.

My pagecontrollers look like this:

Anonymous on Oct 27 2008, 10:03

great article

i agree with you rasmus that many php frameworks give many tools that must be questioned as really useful,

but the truth is that frameworks provide a standard way to do things, instead of lots of antipatterns spread across the project, they help to keep order and maintain visibility (in a certain way)

...but the fact is that layer separation is quite useful,

ive worked in a project where all the code is mixed as a gigantic spaghetti and (we just received the design and it was already made that way) the debugging sessions were a nightmare.....

keep going and good luck!!

Anonymous on Nov 7 2008, 06:50

Thanks Excellent Article!

Very practical and without the scary abstractions of frameworks.

Jquery + PHP rocks

There is a PHP framework that uses this philosophy

<http://code.google.com/p/simple-php-framework/>

Thanks

Anonymous on Nov 16 2008, 07:38

Who could not agree more. After I read the article, I enjoyed to see that all had one common conclusion, simplicity is the aim. It surely confirms what I always have maintained. With most of the new trendy frameworks around and I have seen and checked a lot of them, it was for me an education on how not to write code. Talking PHP/MCV, CodeIgniter seemed to me to be the only one I liked, from the design pattern and the underlying logic as well as the general transparency. But that is my opinion and not comprehensive enough either. Most of the other frameworks I have seen, do seem to indulge themselves in some useless and often oversteered code artistics for whatever reason. I am not saying they are all bad, just too much for the average task. However, one has to point out that most of them around are written for 3rd party users. That is a difference to what is discussed here. Danger with all these overblown framework approach is that one might lose the touch to the real world. Simplicity is the art, the basic code, implementing only what is needed, not what might be needed or you end up where the global markets today: Cul de sac, Unit 00, chapter 11. Because they lost the touch to the people who work for them and the people they call them their customers. Lost in space, global or cyber, what difference does it make. I always remembered a phrase mentioned when I started studying technical engineering in my home town (Saarbrücken, Germany): The aim is to design and to produce as good as necessary, not as good as possible. And that is very much the case in programming. I remember writing so-called comprehensive code, now I write only what and when I need it. And add some more functionality later as needed. I recall so many times waking up and wondering what I was actually writing, realizing that I have lost it, coding far beyond my set target. I believe in OOP, but I keep it reasonable meanwhile. Writing as needed. Aim your target well and don't use a tank to kill a mouse. You will end up in a mess. Thanks to Rasmus for this great article which is after all not just about framework issues, but more about the approach to programming in general.

Anonymous on Nov 23 2008, 17:14

Not only am I late to the party, but I haven't even read through the article in its entirety ... Rasmus's disclaimer at the top made flipping to the comments section irresistible ;-)

It's illuminating to see how vitriolic some people are, tribal even. You'd think Rasmus was a blasphemer or something! I suggest everyone chills out. Take a slug of vodka or something, before you explode.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Anonymous on Nov 26 2008, 12:31

Excellent !!!!!!!!!!!!!

Anonymous on Nov 11 2009, 05:37

I've come across this article a few times now and I keep picking up points each time. I think the comments are some of the most valuable bits so, thanks!

Anonymous on Nov 11 2009, 20:24

I have never been one to delve into OOP. I am sure it has its benefits. I have had some jobs that require some framework. I have been programming since the late 80's and I think straight and basic programming still works.

In the case of CMS', someone came up to me and asked me if i could do his site using one of the leading open source CMS'. I told him, I had my own and would not delve into these open source CMS' (note: I have nothing against them, it's just that I find it easier to do the darn programming myself, especially when a job requires additional procedures. No hassle in skimming through other people's code (and other people's thinking), and no hassles in deployment.)

OOP has its benefits, I agree, but there is totally nothing that OOP can do that procedural, linear or what ever you call cannot do. Some people are fine with OOP, but I am sticking to where life is simpler. :)

Regards to all! and thanks Rasmus for the work you have done and for a fine article.

Anonymous on Nov 29 2009, 17:28

I agree with Rasmus. If you have been developing applications and re-use in future is an issue, you should keep re-usable code blocks for future applications. Trying out every new framework, CMses etc wastes time of learning while it also reduces development in some sense.

It's better to stick to standard php and its techniques with custom libraries instead of trying foreign frameworks.

Doing something with instant knowledge is better and powerful than finding short cuts and put the things on the way without knowing the background.

Anonymous on Dec 21 2009, 20:53

One thing often NOT mentioned with frameworks or 3rd party libs of any kind are the risks of them breaking backwards compatibility or simply ending development and support. I've seen this countless times where version 2.0 breaks 1.x compatibility and sometimes there's not even an upgrade guide. EXT JS lib is a good example of this sort of thing.

I've worked with ruby gems in the past, and pretty much every gem update broke something in the project that required debugging and work arounds. In the sysadmin role, I wanted to just pin gem versions, but you never know when you're dealing with a major security issue or just some random code updates.

PHP is not immune to this phenomena either. There are some PEAR libs that have done this...the HTML_QuickForm package comes to mind. The original is deprecated, but HTML_QuickForm2 is alpha code with NO help in upgrading.

Sometimes those initial conveniences can become long term serious problems.

Anonymous on Dec 23 2009, 02:28

Is the styles.css listing missing? I could not find this listing in this article.

Anonymous on Dec 24 2009, 17:21

Nice article. I note the pseudo-academics with their so called "elegant" OO techniques and code reusability look on in disdain. As we say in the UK it's "horses for courses"...

Many many HUGE sites have been built around php and NO MVC framework (shock horror!)...

Lets be frank - we did well before them... and we'll do well even if we don't use them...

regards

Anonymous on Dec 26 2009, 22:35

Thank you for this beautiful article.

Anonymous on Dec 28 2009, 05:45

Great article (as evidenced by 3 years of responses) I'm surprised at the extreme polarity of them but strong opinions help illustrate subtle differences so I guess that's a good thing for helping others see the possibilities.

Ultimately I took it as a clean demonstration of square/rectangle logic to the Framework/MVC. A framework is [usually] an MVC but an MVC need not be a framework.

As apparent as it may be to most, I think it is enlightening to many that you can adhere to MVC patterns without doing so on top of a framework. This is very helpful for smaller 'shops' grappling with the decision to adopt a framework vs. just adopting more structured coding conventions.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

This in no way defames the frameworks which are indispensable to many larger operations.
Anonymous on Dec 28 2009, 12:21

I'd really like to see this example go further. I love the idea, I like developing in frameworks but have found that a lot of work needs to be done to make many of them fast enough to actually be considered production. I'd like to see an example that goes more in depth. Example - 2 or 3 controllers, models, and 10 or so views. Something that shows that as it grows it's still "easy" to keep things organized. Also something that shows use of "templates" like you've done that are a little more advanced.

Again, I love the idea.
Anonymous on Dec 31 2009, 13:18

Hi I have found a very nice article that explains how MVC can be used and other bloating features of a framework can be eradicated to form a no-framework. Really interesting reads.

<http://www.devlopr.com/do-frameworks-fail-when/>

<http://www.devlopr.com/after-all-what-is-this-no-framework/>

<http://www.devlopr.com/the-mystery-of-the-dead-project-%E2%80%93-exposed/>
Anonymous on Jan 2 2010, 21:33

"You will notice a distinct lack of input filtering yet if you try to inject any sort of XSS it won't work. This is because I am using the pecl/filter extension to automatically sanitize all user data for me"

Does this mean you have your ini setting
filter.default "string"
Anonymous on Jan 16 2010, 10:25

I believe it is always going to be more efficient and transparent to build a framework for a given application, rather than building an application from a framework.
As an analogy, you wouldn't build a house from an outline of a skyscraper, well you could but people may think you're wasteful and a little crazy.
Anonymous on Jan 26 2010, 16:02

Employing a framework in a language is not so different from switching from a low level lang to a higher level one. For example, most people do not code web apps in C++ (not the front end anyway). I know one guy who does, but there's plenty of evidence that shows you don't gain much performance back compared to a scripting language such as perl, python, and of course php. The bottlenecks usually lie elsewhere (ahem database).

Using php means you don't have to write a session handler from scratch or various other important features needed for real web development. Some languages can't do much web work without a framework. Ruby and python have zero built-ins for web dev and without a framework, you're rolling a lot of core stuff yourself. With php, that is not the case, and there is some memory overhead to loading all the core functions and classes. There are times I wish I could control what gets loaded in and what doesn't. It would be amazing if you could load everything in dev mode, run some analysis on required libs, and then explicitly declare your imports during production. How cool would that be? Oops I'm wandering...

Anyway, the framework approach is very anti-posix. The beauty and key to the *nix principle is having many small programs that do one thing just right and the ability to chain things together for power and flexibility. Frameworks take the monolithic approach, the end result being bloat and loss of control. It's the same thing I hear C++ guys say about scripting languages and even java. Tradeoffs. In a language like php, frameworks seem extravagant, unnecessary, and just plain restrictive. Better to take the piecemeal approach and include just what you need. This is what pear/pecl is all about. Unfortunately pear quality and maintainence has been quite poor to the point where I trust very few pear packages these days.

A php "framework" that lets you pick and choose components to use within your design structure rather than force structure upon you would suit php best. And isn't that pretty much what 3rd party libs were before the framework fad came about?
Anonymous on Jan 26 2010, 18:50

Hello!

I've been implementing my own mvc in php recently and stumbled upon this, which is in many ways similar to what I've been writing. I have a concern about the controller, which i am facing right now in my code.

When I look at your code, it seems like the real controller is in common.js's usage of event listeners in fancyItems(). Initial page load is controlled by the thing you call the controller: add_c. I understand that in conventional mvc definition, your add_c is the controller. But really add_c does nothing more than what back-end model should be doing anyway. The view never directly interacts with the model in your example; it interacts only with your add_c controller. So I consider add_c, what you call the controller, really a layer upon the model. The view is defined by your html, but more significantly it is defined by the javascript. AND, in my opinion, it looks like the javascript also functions as your controller. (My purist view thinks "controller" refers to the thing that processes user input and decides what to do with it, "view" referring to the output and how to serve it.)

Do you see it this way, too, or could you? In my application, I have taken to using asyc requests to update the view (and I use YUI 2.7), and my javascript ends up doing what you do--it passes some parameters and a resource (URL-type thing) to the back-end code, which figures out which back-end controller (similar to your add_c) and view (similar to your add.php) to give control to. I can see flaws with this.. it just doesn't seem as beautiful as I imagined my mvc would be. Of course, I am willing to incorporate some imperfections, but only if they are totally necessary.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

I guess my concern is really highlighting a flaw I see in the way mvc is being defined/interpreted/implemented... Maybe it would be useful to separate client-side view and controller functionality. So the callback function `fn()` would be seen more as a view updater, the `clickFormEvent()` more as the controller. Or if not separate them, at least begin to think of them in this way, so that when the application is expanded and more things are added, you might consider more of a separation. Any thoughts?

Anonymous on Feb 12 2010, 16:05

Like retry said "A php 'framework' that lets you pick and choose components to use within your design structure rather than force structure upon you would suit php best. And isn't that pretty much what 3rd party libs were before the framework fad came about?"

I think that will make a lot of sense for PHP, I think like having a YUI library approach, but for PHP. That would be really cool!!

Anonymous on Feb 23 2010, 20:30

@Jose/@retryistherealretry

I think Ruby's Sinatra and PHP's rough equivalent - Fat-Free Framework (<http://fatfree.sourceforge.net>) - are both in the right direction. Programmer has freedom in how MVC design pattern should be separated, or even no separation at all. No clutter in directory structures. It implements the front controller design pattern just as well, so you have all routers/controllers in a single file. But even that is under your control.

All told, it could qualify as a no-framework framework. It's not the size, the speed, the OOP or procedural style employed, blah, blah, that matters. It's how elegant the code is, how the code is easily distributed across a team, and how usable it is.

Anonymous on Feb 25 2010, 05:51

I agree in most part with the author, but I think there are some obvious points that should be noted, and why abstraction can quite often end up saving you time...

I think this kind of style and approach is great for simple projects, but the moment you need to implement themes, offer support for different storage mediums.etc, you suddenly feel "stuck" and have to start coding "around" the code, than with it.

My templating engine, PHLite (look it up on github), uses this philosophy of simplicity.

All in all, I think this post is a great one and definitely a great brain exercise. i.e. How CAN I make my code simpler?

Cheers =)

Anonymous on Mar 24 2010, 19:52