

Re: The structure of PHP/Web Application coding.

Re: The structure of PHP/Web Application coding.

Source: <http://coding.derkeiler.com/Archive/PHP/comp.lang.php/2007-07/msg00018.html>

- *From:* "Peter D." <peterdi@xxxxxxxx>
 - *Date:* Sun, 01 Jul 2007 15:46:35 -0000
-

On Jul 1, 11:23 am, Henk verhoeven <news1@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

Hi Peter,

From your mentioning of Design Patterns i assume you want to/are practicing OOP. By far the most of my experience is with OOP, so i need to make this assumption to think that my reply is relevant.

Peter D. wrote:

I have been programming PHP for a while now and always seem to run into the same problem when working on more than trivial apps. Most of my coding is for personal projects anyway so it really isn't a big deal but I hopefully plan on doing more serious coding in the future.

My problem is that as I code and the app grows ever larger, I always feel like I am digging myself a hole I can't get out of in the end. My code is alright, readable and not too bad to debug but I always get that feeling like I should be re-considering my design choices now or I might really be in trouble later.

Do you guys ever get to that point? If not, how do you avoid it?

Certainly! And even with 14 years of experience with OOP and a strong focus on design & reusable code, i don't feel ashamed. A little bit of History: Much of OOP as we know it was invented during the development of Smalltalk. Smalltalk was strongly influenced by ideas from cognitive psychology and was not only a programming language, but also a concept and technique of user interfacing, an IDE, and a new approach to programming as a task for humans. Other programming languages (and Graphical Operating Systems too) have since the 80's been cherry-picking from Smalltalk, but it still seems not commonly known how brilliant and ahead of its time Smalltalk has been.

Re: The structure of PHP/Web Application coding.

Some of the leading old school Smalltalk developers have extended the Smalltalk approach to programming into a more complete and explicit method: Extreme Programming (XP). One of the most controversial guidelines of this method is not to think far ahead in your design but instead, build "the simplest thing that could possibly work" for your current purposes.

Of course this can not lead to reliable and maintainable software without a second guideline, that balances the first: "Refactor relentlessly". Refactoring is improving the design of existing code. This is not something to do after several months of coding. Refactoring should start almost immediately from the start of any project and is to be done so frequently that it comes close to being done "all the time".

As extreme the XP guideline of "the simplest thing that could possibly work" is in terms of designing ahead, it is in terms of the design itself: "the simplest" is defined as "once and only once". Basically if you have any piece of code that is the same, or very similar to any other piece of code, you need to refactor it into one piece of code that is reused for both tasks.

Of course reuse has its limitations, sometimes it just makes things too complex, so in practice you always have to make compromises. But the message remains the same: You need to re-consider your design choices *much much sooner* to avoid getting into trouble. A design is only a temporal optimum. If you add some code, the optimum will change. That is not something to be avoided. There is no need to be ashamed about it. It just needs to be done right away, and not to be postponed until it is a big task. The hunt for simplicity is always on, relentlessly!

I know that with the right amount of planning anything can be accomplished in a reasonable amount of code/time. I guess what I really want to know is if there is any kind of code strategies that work well when designing an application.

For more info on refactoring, you may read the well-known book "Refactoring" of Martin Fowler.

For more info on Extreme Programming, you may read "eXtreme Programming eXplained, Embrace Change" from Kent Beck.

Of course the design depends on the application. But if you are developing similar applications as I have been doing, you may consider using a framework. IMHO a framework is simply the outcome of applying the principles of XP over the development of several similar applications. Within a single project the principle of "once and only

Re: The structure of PHP/Web Application coding.

once" will already leads to some code being heavily reused. If you try to use that code in a second project, it will inevitably need more refactoring. I have been doing that refactoring in such a way that part of the code can be used in both projects. I have repeated that process over several projects. And i have repeated the entire process of building frameworks three times, each time starting from scatch, but reusing the designs of the previous generation, but also throwing out as much as possible to make the simpelest thing that could possibly work.

Some XP proponents will not like this idea, they will point to the principle of "travel light" and say that the frameworks make you carry around a lot of unecessary (complex) code, that's not necessary and does not need to be so complex for the application at hand. I agree, but i have several reasons to prefer using the framework over starting from scratch every time:

1. With the framework i have a starting point, a basic design for the application that is based on practical experience from former applications i refactored the framework from. Yes, i loose time on complexity, but i win time on doing sort-of the right thing.
2. In the early stages of application development the framework has proven to be adding most to my productivity. In fact the first prototype will be available ten to twenty times sooner then when i have to start from scatch. That means earlier customer feedback. Earlier feedback means earlier clarification and maturing of requirements. That means less time spent on building the wrong things, and *much* less risk of running out of time becuae of that.
3. In the later stages it may occur that some parts of the application are not similar enough to profit from the framework. That's no point, having the framework does not stop me in any way from building them from scratch.
4. The choice of XP to start simple and refactor later is based on the assumption that the investment of the software community in better tools, practices etc. has payed of, resulting in the costs of change not rising dramatically over time, but instead flattening out. What if the investments in frameworks are doing just that? Traveling light is a good thing who would refuse to use a car when having one and the roads are clear, just because the car is adding a lot of weight?

I've read a lot about Design Patterns (Factory, AbstractFactory) and I just can't seem to think far ahead enough in my code to be able to implement these patterns succesfully.

To be honest, i never tried to implement a pattern. It just happens that when i am looking for "similar code" that i may eliminate, these patterns help to find it and give directions on how to get rid of it.

Re: The structure of PHP/Web Application coding.

On the other hand, when i am not looking for patterns, my code often tends to evolve into something similar to some patterns anyway. Simply because the problem i am trying to solve is similar and the pattern is the simplest solution.

But in the end, i do want to get rid of the patterns as well. In the end, patterns that occur several times in the same applications often can be refactored so that the pattern is only on one place...

Are there any sites which clearly give examples on how to design medium to large applications that are scalable/simple... or am I just still too new to the game and just need to practise my coding a lot more?

Maybe you just need practice examples. In Smalltalk we had the class library and the code of the IDE itself we could learn from. Not always brilliant code and not really a solution on uncovered grounds like persistency, object caching, transactions etc. but it was a start. Most Smalltalks also offered a powerful IDE for browsing and searching and did not keep much code 'out of sight' to application developers. PHP does not include a class library, but there is PEAR, and there are several good open source frameworks available.

I think that you can learn a lot from using such a framework. Use it as a white box, not a black box: when building your applications, study the code of the framework, look for undocumented features, for ways to write less application code by using some framework code in undocumented ways. In the beginning this will slow you down. Or more correct: it will slow down your progress with developing your applications, but speed up your learning. If learning happens to be what you want, that's time well spent!

Greetings,

Henk Verhoeven, www.phpPeanuts.org.

phpPeanuts is a framework for developing business applications in php. On the website the main tasks of application development with the framework are documented and demonstrated in tutorials that come with working code examples. The use and study of the framework as a white box is supported by the hypercode browsers, that allow you to browse and search the framework and examples code on line.

N.B. From the above you may conclude that the better frameworks are relatively small compared to the functionality they offer. They only offer generic functionality and allow/expect application developers to add application specific functionality without having to modify the

Re: The structure of PHP/Web Application coding.

framework itself.

Thank you for your answers.

I have been looking into a PHP framework called symfony for the last couple weeks and have been enjoying the learning experience a lot. I think I just need to look at more code and just read a lot more.

Thank you again for your help.

Peter

.