15 Important Considerations for Choosing A Web Dev Framework

by Siddharth Dec 7, 2009

Read Time: 11 mins Languages: English

New web development frameworks are sprouting out at a more rapid pace than anyone could keep up with. In this article, we are going to determine how to decide on a framework for creating your next, hot web application.

In this current day and age, pushing out a finished, polished application well before your competitor is key. Coding everything from scratch, excluding even the mundane things, can be extremely time consuming and makes the developer spend time reinventing the wheel, time which would rather be spent implementing new features or tightening up the code base. This is where web development frameworks come in. They often cover all the usual aspects of an application including database access, authentication, session management and much more.

Today, we are going to take a look at the various aspects you should be concerned about before choosing a framework. Interested? Let's get started right away!

1. Usage Context

Before you even start looking at a frameworks, you'll need to make a list of your requirements and whether a framework is suitable for that purpose.

You are in need of a framework if:

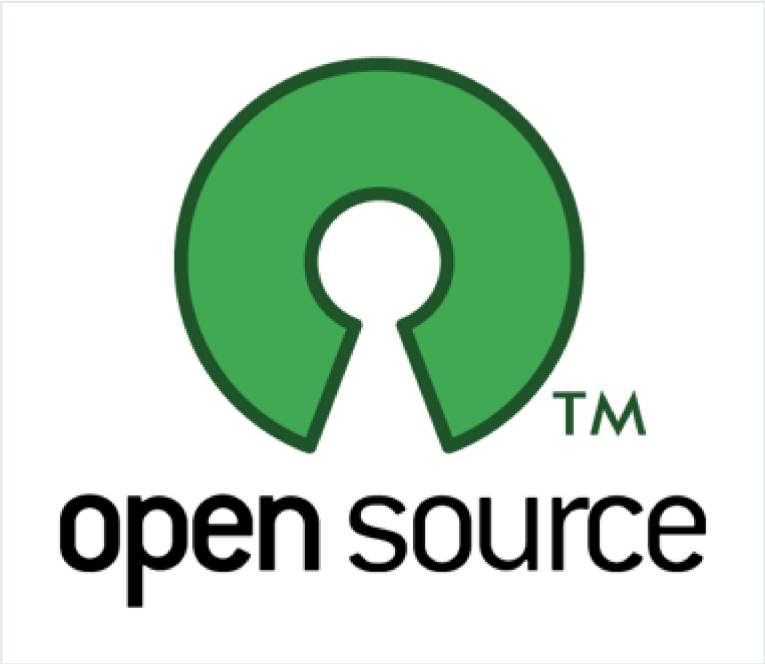
- your application is primarily based on CRUD operations
- you need proper separation of the UI and underlying logic but don't have the time to implement a proper system
- you find yourself having a self made library you use in each of your applications covering user authentication, sessions and other usual operations associated with creating a web application
- you have a boss who wants you to create a CMS for them in 2 days and you already know the framework

You DON'T need a framework if:

- you want a pretty URL system alone
- you want only a specific part of the framework like its ORM
- you are on a tight timeline and you need to learn the framework from

scratch
- you were told frameworks cure cancer

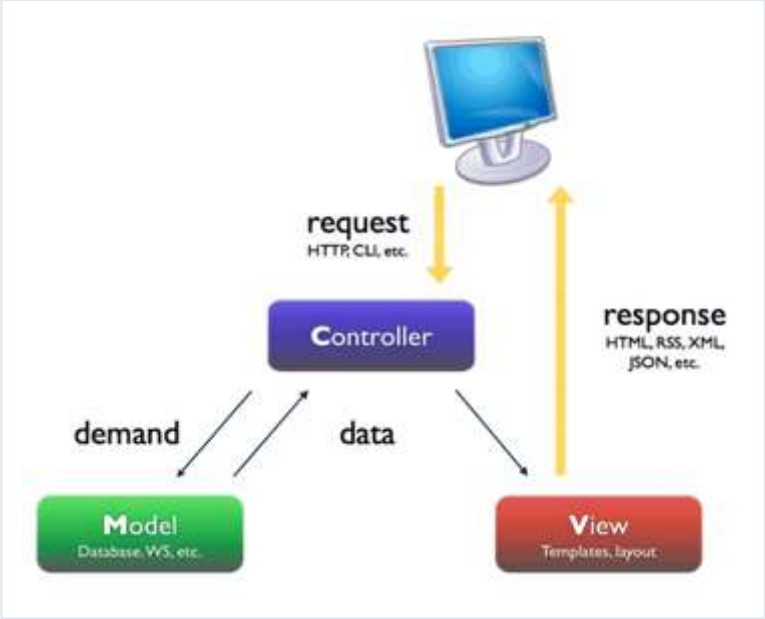
2. License



Before you start developing with a framework, see what kind of license the framework is distributed under. While most licenses are pretty liberal to work with and lets you create commercial applications, some of them are not so generous. The last thing you'd want is to create an entire application only to find out that the license doesn't allow you to distribute it commercially. Better do your research before than suffering after.

Do note that this isn't limited to the framework alone. The plugin or extension you made use of for the extra functionality may have a hidden clause. Check it's license too!

3. Software Pattern



From the Symfony project
Almost every framework out there exclusively uses the MVC pattern. MVC, which stands for Model-View-Controller, helps you keep your data: the model, the logic: the controller and the user interface: the view, separate from each other. This in turn lets you write better, tighter code which ultimately results in better applications.

Just because almost everyone uses MVC doesn't mean that is everything you need to know though. There are a couple of variants including MVP: Model-View-Presenter, MVA: Model-View-Adapter and AVC: Application-View-Controller.

4. Hosting Requirements

As web developers, we may be inclined to building applications on cutting edge platforms but often the need and budget of the client comes first. Often it may be out of the budget to get a dedicated host to place our applications on and we'll have to settle with shared hosting with normal modules and settings.

Frameworks which play nice with shared hosting include:

- CodeIgniter
- CakePHP
- Kohana
- Zend Framework
- Most other PHP frameworks

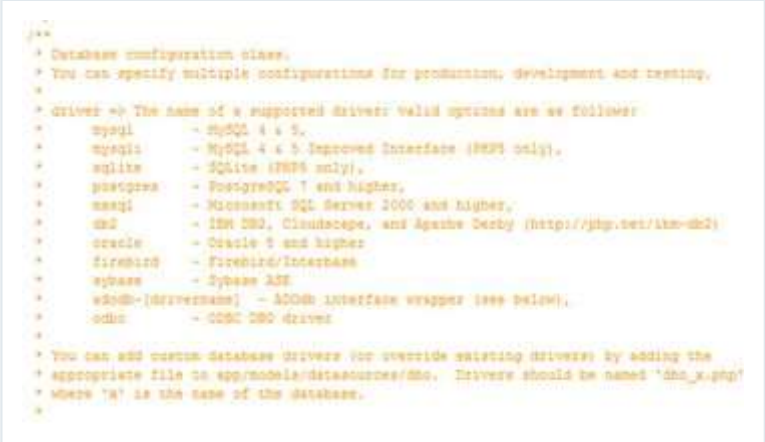
Frameworks which require relatively non-traditional setups:

- Ruby on Rails
- Django
- Pylons
- Most non-PHP frameworks

In all honesty though, you can still run frameworks like Django off an off the shelf

In all honesty though, you can still run frameworks like Django on an off the shelf shared host. It'll just require that the server has the necessary module installed. You may be able to run it off CGI but it'll be a lot slower than running it natively.

5. Ease of Installation

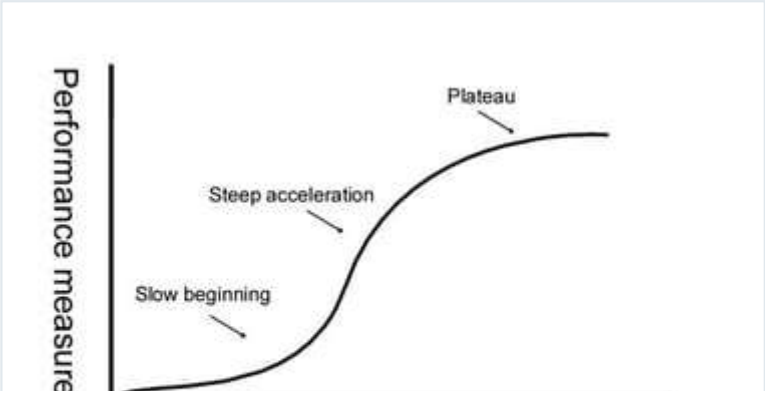


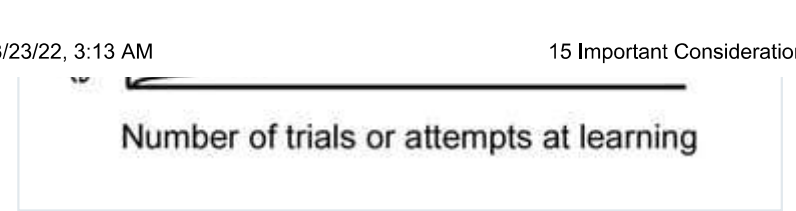
Ease of installation plays a very important role whilst choosing a framework. A framework, however feature laden or quick it may be, can pose quite a problem if one has to run through a number of steps just to get it installed and working.

This will also pose a big problem once the application is ready, tested and needs to be deployed to the production server. A framework with ease of installation and just as easy deployment gets brownie points here.

For a lot of the frameworks, set up is as simple as setting the right values in the configuration file whilst for others it may be a very time consuming, elaborate affair. Choose a framework which lets you get up and running as rapidly as possible.

6. Learning Curve





Every framework has its own tiny universe: naming conventions, directory structure and what nots. Some frameworks are rather flexible when it comes to these while others are very, very strict throwing up errors at the tiniest of mistakes. Some frameworks follow general conventions while implementing a feature while others may strike out and do its own thing. While choosing a framework, remember to choose one that has the smallest possible learning curve.

If you don't know the language the framework is written in, make a note of including the language itself to the learning curve. I've seen a number of developers jumping to Django from CakePHP and struggling since they need to learn both Python and Django at the same time. If you need to learn both the framework and the language it was written in, pace yourself!

7. Core Library

```
def create
  cookies.delete :auth_token
  # protects against session fixation attacks, wreaks havoc with
  # request forgery protection.
  # uncomment at your own risk
  # reset_session
  @user = User.new(params[:user])
  @user.save
  if @user.errors.empty?
    self.current_user = @user
    redirect_back_or_default('/')
    flash[:notice] = "Thanks for signing up!"
  else
    render :action => 'new'
  end
end
```

Let's face it; it is for the core library that most people adopt a framework. The library must be in such as that it frees you from writing repetitive code but still provides a way for you to tinker with it if you need more control or features.

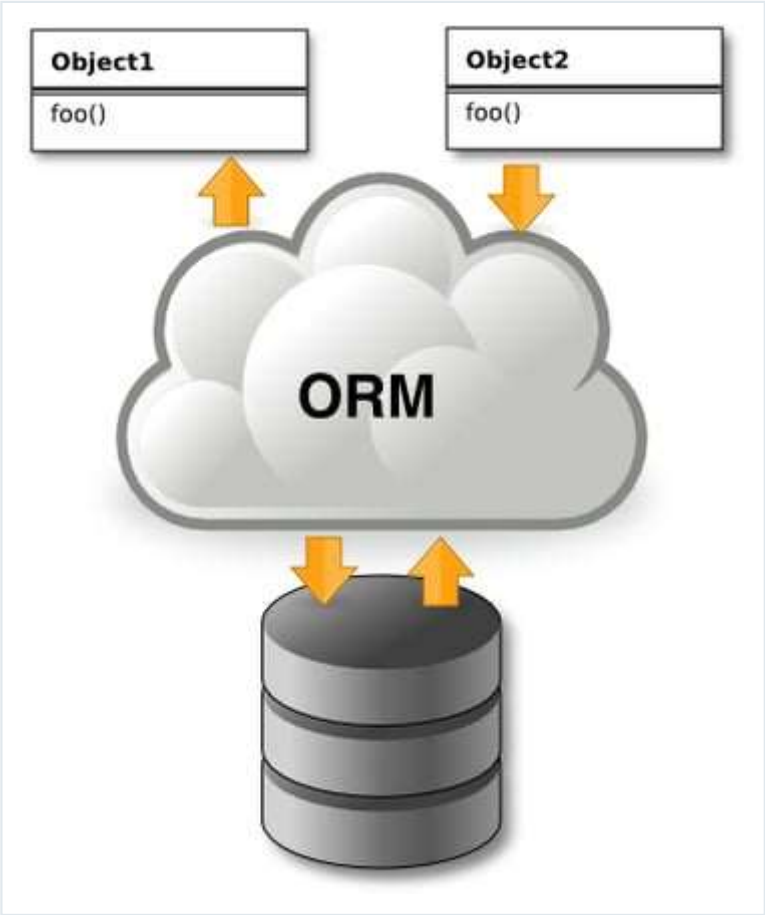
Most frameworks offer libraries which cover almost all of the following list:

- AJAX
- Authentication
- Authorization
- Caching
- Data Sanitization
- Data Validation
- Templating
- URL mapping or rewriting

Of course, not every one needs a framework brimming with features. A lot of people prefer frameworks to handle the bare minimum whilst letting the developer handle the rest. In these cases, you'll need to make sure the framework in question has only the features you need.

A current trend among frameworks is that they are created as a library of libraries. In other words, it lets you swap out parts of the library with another part of your choice. An excellent example of this would be Pylons. It lets you switch out almost all of its parts starting from the ORM right down to its templating language. People like these loosely coupled frameworks just as much as the frameworks which are tightly couple with respect to its core components.

8. DB Abstraction and ORM



Almost every application has to access a database to either read data off or edit its contents. Either way, you'll be doing this along the length of the application and with this in mind, most frameworks lets you use an database access class you can make a use of. So while choosing an application, choose one which lets your application become database agnostic. You'll never have to care about the database part incase you need to switch out databases if your framework takes care of that.

The second part you'll need to think about is the framework's ORM capabilities. Without getting technical, ORM or Object Relational Mapping lets you express data as an object and see how it relates to other objects. Imagine an object database you can pull information from, if you will.

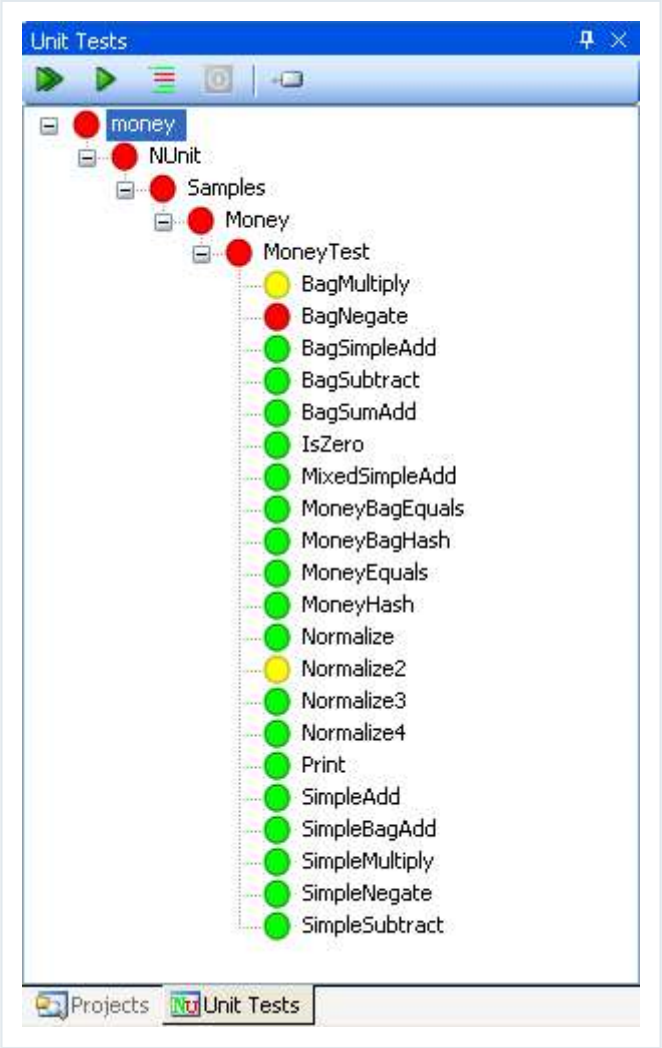
Frameworks which have ORM capabilities include CakePHP, Django and Ruby. With frameworks like Pylons, you can use your ORM of choice.

9. Included JS Library

Another point of contention is the bundled JavaScript library. While most libraries let you swap out the library with ease, the AJAX methods inside the framework are mostly still aimed towards a specific JS library. This implicitly means that you'll have to manually write the functionality yourself. On the other hand are frameworks with library agnostic methods which let you swap out the JavaScript library with little to no hassle.

Just as a reference, both CakePHP and Ruby on Rails ship with Prototype and Scriptaculous as standard.

10. Unit Testing

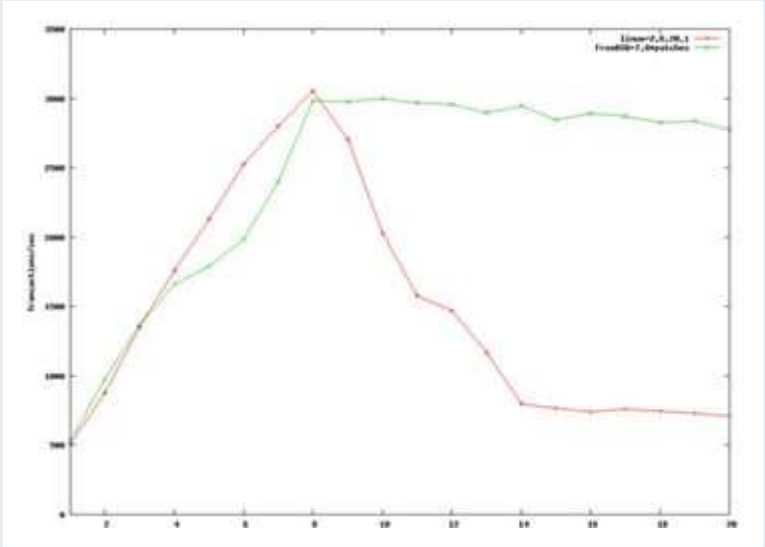


I am one of those developers who swear by unit testing. Wikipedia defines unit testing as:

Unit testing is a software verification and validation method in which a programmer tests if individual units of source code are fit for use. A unit is the smallest testable part of an application.

Frameworks which lets me write unit tests are a definite plus in this case. A lot of frameworks including Code Igniter, CakePHP and Zend lets you create custom tests in addition to the core tests to check the critical parts of your application.

11. Scaling



The average web developer doesn't have to worry about the scalability of a framework. As often is the case, I/O and network latency are often the issue rather than a framework's scalability. Even Twitter's mythical scalability problem wasn't the fault of the framework in question.

If someone asks you to layoff a framework pointing out *scaling problems,* disregard them. The framework is rarely the reason of the scaling issue. Sure, you could optimize the code a little bit but often the brunt of the scaling issue lies elsewhere.

12. Documentation

A framework's documentation is often key to its success. Well explained, detailed documentation draws in the power users and evangelists who then bring in more people. With shoddy, confusing documentation people are going to just walk off confused and annoyed.

Look for a framework which has thorough documentation with plenty of examples, snippets, sample code, articles and tutorials. Screencasts like Jeffrey's are a special plus since it'll let you zip right along.

13. The Community

Inevitably, even with proper documentation, you are going to run into errors to

Inevitably, even with proper documentation, you are going to run into errors to rectify which, you are going to have to ask the community behind the framework for help. I've personally interacted with communities which vitriolically attacked programmers new to the framework and sneered at them while on the other

extreme I've seen communities cheerfully welcomes newbies and teach them the tricks of the trade. I think there is no need to say which framework I ended up choosing and started working permanently on.

As often is the case, the communities behind a framework make or break the framework. Too snobby and you'll resent the framework instead of the people. Well mannered and you'll gravitate towards the framework. Choose a framework which has a friendly community which helps developers new to the platform.

14. Bug Fixes/ Updates

One of the reasons web developers shy away from just creating their own frameworks is the fact that they alone are in charge of fixing the bugs and updates. With a big framework, you literally have thousands of programmers sifting through the code and putting it through its paces on a daily basis. Bugs, if and when they are found, are squashed as soon as possible after they are found.

Choose a framework which isn't stagnant. You don't want a hacker to tell you that a security vulnerability exists in the framework through a page he hacked on your site. You'd rather hear that from the framework developers, hopefully with a link to a patch to the issue. Choose a framework which is updated often, is open

about the bugs it finds and more importantly fixes the bugs people come across as soon as possible.

15. Ease of Creating an Extension and Availability

While a framework covers all of the important bases of an application, chances are, you'll still have to write a bunch of code. Make it generic enough and you can re-purpose it into a component suitable for reuse in your other applications or even better release it to the general public so they make use of it in their applications..

Choose a framework which lets you extend the framework easily and with minimal fuss. With CakePHP for example, extending a controller is taken care of by *components* and views by *helpers*. In either case, creating an extension is as simple as defining a new class which inherits from a parent base class.

While choosing a framework, also keep in mind the availability of plugins. Often you won't have the time to create a custom extension from scratch. Having a huge pool of extensions to choose from greatly alleviates this issue. Choose not by the number of extensions but by the quality of the extensions.

Conclusion

And we are done! We looked at all the aspects you should consider before choosing a web development framework. We looked at everything from mapping out whether it'd suit our purposes to bug fixes and updates. Hopefully, this has

been useful to you and you found it interesting.

Questions? Nice things to say? Criticisms? Hit the comments section and leave me a comment. Happy coding!

Follow us on **Twitter**, or subscribe to the **Nettuts+ RSS Feed** for the best web development tutorials on the web.

Web Development Frameworks

Did you find this post useful?

 Yes  No

Siddharth

N/A

ENVATO TUTS+ +

JOIN OUR COMMUNITY +

HELP +

tuts+

30,009 Tutorials | 1,316 Courses | 50,078 Translations

Certified B Corporation