Laravel

**1. Lack of continuation between versions**

There is no consistent change from Laravel 4 to 5. On the off chance that you attempt to refresh the code, it could break the application.

**2. Some upgrades might be problematic**

This isn’t exclusively a Laravel issue, however, PHP frameworks do demonstrate issues for long haul bolster forms as the redesigns may turn risky. So the developers are encouraged to play it safe before overhauling a versatile application/website.

**3. Often fails to provide mobile app richness**

Full-page reloads can be somewhat substantial in mobile applications when contrasted with websites. In such cases, web developers will, in general, utilize the framework as backend JSON API as it were.

Django

**#1 – Regex to specify its URL:**

You can create simple and easy-to-read URLs. However, Django uses regex to specify its URL routing patterns and that makes the code larger and creates complicated syntaxes. This is an example straight from the documentation:

**#2 – While maintaining backwards compatibility, it's moving far too slowly**

The framework has dedicated itself to backwards compatibility.

**#3 – Is Django too monolithic?**

For sure, and this is the reason why Django maintains large, tightly-coupled product. The framework is monolithic and it pushes you into given patterns, but it is more fun when you can program yourself — choosing architecture, structure and design patterns.

**#4 – It feels like it has too much software for minor projects**

**#5 – Template errors fail silently by default**

It seems like the framework developers didn't pay attention to mistakes when they stepped to their own class-based views. They are extended via inheritance, that means it will make everyone’s life entangled in the subsequent versions.

**#6 – A process only handles a single request at a time**

Unluckily, WSGI-based servers cannot be utilized to develop real-time apps, as WSGI protocol is synchronous. WSGI server can handle only one request at a time.

Ruby on Rails

**Runtime Speed** — The most cited argument against Ruby on Rails is that it’s “slow”. We would agree, certainly when compared to the runtime speed of NodeJS or GoLang. Though in reality, the performance of a Ruby application is incredibly unlikely to be a bottleneck for a business.

**Boot Speed** — The main frustration we hear from developers working in Rails is the boot speed of the Rails framework. Depending on the number of gem dependencies and files, it can take a significant amount of time to start, which can hinder developer performance.

**Documentation** — It can be hard to find good documentation. Particularly for the less popular gems and for libraries which make heavy use of mixins (which is most of Rails).

**Multi-Threading** — Rails supports multi-threading, though some of the IO libraries do not, as they keep hold of the GIL (Global Interpreter Lock). This means if you’re not careful, requests will get queued up behind the active request and can introduce performance issues.

**Active Record** — AR is used heavily within the Ruby on Rails world and is a hard dependency for many of the Ruby Gems.

Express.js

**Θ Performance bottlenecks with heavy computation tasks**

**Θ Callback hell issue**

**Θ Immaturity of tooling**

**Θ Growing demand for experienced professionals**

Flask

**Not easy to scale**

**Easier to write bad code**

Since Flask follows few conventions, it’s easy for inexperienced developers to write code that can make your codebase confusing and potentially harder to maintain.

**Few bundled tools**

Flask doesn’t come bundled with a huge amount of tools, this means you might have to build certain features for yourself, or seek out 3rd party libraries, which can vary significantly in their reliability.

Spring

**i. Complexity of Spring**

The major criticism that this framework has seen is that it is too complex and it lacks clear focus as it has more than 2400 classes with 49 other tools which make it complicated for the developers to use it.

**ii. High Learning Curve**

If you are a new developer in this area it would be quite difficult to learn Spring Framework. This is due to new programming methods and the detailing each of them requires for developing the application.

**iii. Tons of Parallel Mechanisms**

This point is also mentioned in advantages section, but this can be disadvantageous to given that it can give developers lots of options and that causes confusion.

**iv. Lots of XML in Spring**

If you ever worked with Spring framework you will get to know that developing Spring application requires lots of XML.

**v. Lack of Guidelines**

There are no clear guidelines on several topics for the developers in Spring documentation such as cross-site scripting attacks or cross-site request forgery attacks. Also, it has several security holes.

.Net Core

 .NET Core is not fully matured and is still evolving.

 WCF, WPF and Window Forms support is still being added to .NET Core.

 .NET Core does not have all the functionalities which are present in .NET Framework so far. Obviously, this will change in coming versions.

 Some 3rd party libraries are not supported by the .NET core so in those cases you need .NET framework.

 Feature such as Code access Security are not present in .NET core so .NET framework do have the edge in that case.

 You cannot use Xamarin with .NET Core at this point.

 Even though .NET Core supports VB and F# but that is not supported for all the projects types.

 In case we already have an application in .NET framework then no need to migrate the existing application to .NET Core.

 The material, documentation and tools available for .NET Framework are far more and more matured. compared to .NET Core. Even though .NET Core is catching up fast but there's still a long way to go.

 All API's of .NET Core are not tested and mature as they are very new compared to .NET Framework.

 .NET Core is still new and evolving. There will be many changes in .NET Core.

 There are not many jobs in the market for .NET Core developer

Golang

**It isn’t a great general-purpose language.** Have you ever had to open a can with a philip screwdriver or a beer bottle with a lighter (oh, yes, my friend could do that) because of the lack of a better tool? Go is similar.

**Go asynchrony isn’t easy.** Who ever said channels and goroutines are trivial, they were almost always comparing them to threads.

**Nil and multiple return values.** I don’t think there’s any need to mention this. People have said enough about null pointer.

**Go’s type abstraction is very primitive.** This is by design since more abstraction means more overhead. However, this means you’ll end up with lots of boilerplate code.