Case Study Report

On

Comparison of spring and other frameworks

Submitted by

Jitu Khubchandani

(**PRN** : 130301420**35**)

Prabhu Narayanpethkar

(**PRN** : 130301420**75**)

Table of Content

Topic **Page No.**

1. Introduction .... (1)
2. Need of Web Frameworks.......... (1)
3. Developer Productivity Report.. (2)
4. Rapid Application prototyping… (2)
5. Framework Complexity.............. (3)
6. Ease to Use................................. (5)
7. Throughput/Scalability.............. (6)
8. UX, Look and Feel....................... (7)
9. Summery...................................... (8)

1. Introduction

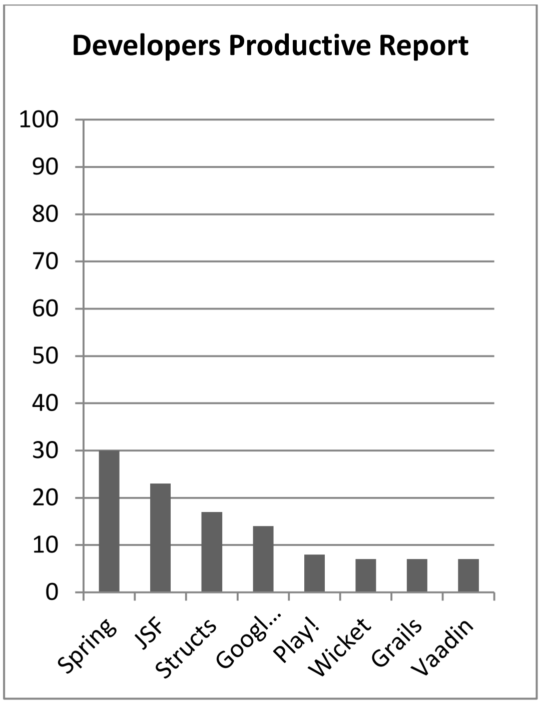
Web Frameworks are all very different and have typically been created for different reasons and to achieve different goals. Which Java Web Framework will you use in your next project and why would you choose one over the other? There are many features which may sway your decision and of course it will depend on the type of app you're building.

2. Why do we need web Frameworks?

Well, coding good-looking web applications in Java isn't super easy at time. It can be hard to do and mostly doesn't give us the rich front-end we strive to deliver to happy users. This is really the catalyst which has caused Web Frameworks to be created. Both, functional and non-functional web app requirements have led to the need for various web frameworks to be created, but this brings us to the opposite problem... there are so many choices out there to use, which web framework should you pick for your next web app?

3. Developer Productivity Report

Developer Productivity Report and see what we had in their back in 2012. According to over 1800 developer responses, here's what we found



More than just looking at market share and usage in place, we wanted to extend this report on Java Web Frameworks to look deeper at these eight libraries, and find out about what is really important to developers, in addition to looking into how different frameworks make sense for different use cases.

This report is the first of two and will focus on a feature comparison across the following categories:

- **Rapid application prototyping**
- **Framework Complexity**
- **Ease of Use**
- **Throughput/Scalability**
- **UX, Look and fee**

4. Rapid Application Prototyping

Whether you're using the framework for the first time, or you're an expert in all things framework related, it's important to be able to develop quickly to prototype an idea or just try new things out. This section rates each framework for the ability to produce content from scratch with rapid results.

SPRING MVC

If you're looking for a framework to help you generate a application fast and clean, Spring really shouldn't be your go to. It's MASSIVE and hard to grasp if you're just starting out. For a quick template, you can always download the Pet clinic package and strip all of the unnecessary stuff out - but even that will take time, and you need to be able to tell what is necessary.

Spring Roo, a subproject of spring that supports Spring Framework, Spring Security and Spring Web Flow, is spring's convention-over-configuration solution for rapidly building applications. It focuses on Java platform productivity, usability, runtime avoidance, lock-in avoidance, and extensibility via add-ons. It's in the process of being expanded and has a lot of potential.

Reason: Much preexisting spring knowledge is needed. Plain JSP & Controllers do not provide out-of -the-box components and widgets that can be used.

GRAILS

Grails' simplicity rocks. Next time we need to implement some small-to-medium CRUD application we'll consider using Grails. The setup is very fast and scaffolding (code generation) saves a lot of time. Convention over configuration principle helps you to forget almost all of the configuration hassle.

Grails comes with a reloading mechanism out of the box, but it has some limitations (like it can only reload Groovy classes) and you may still want to use [Blatant Product Pitch] JRebel if a part of your project is in Java

Reason: Top marks here. Scaffolding, conventions and hot code reloading, what else could you want?

JSF

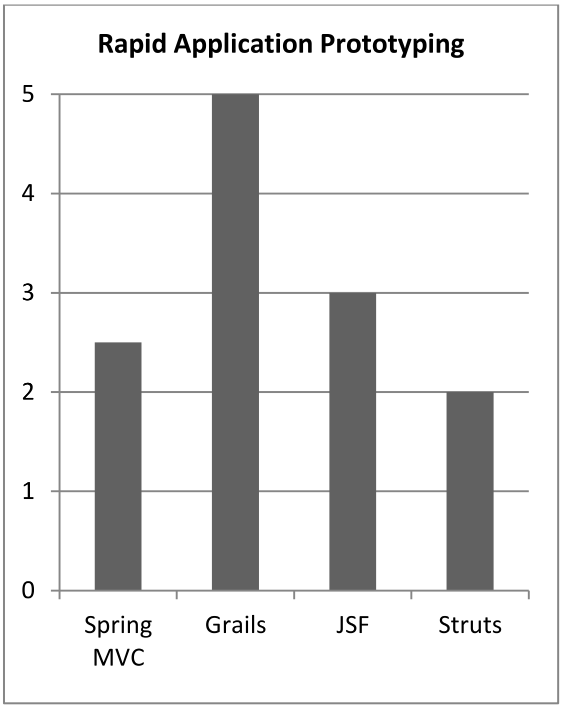
JSF is not fantastic for quick prototyping; code generation is not a built-in feature and prototype applications require just as much configuration as a full application. This is not really JSF's fault, as it is reliant on the Java EE specification. JSF does have several useful Maven archetypes, however, that do provide a good starting point for a basic application. Prototyping can also be achieved with the vast array of sample projects available both online and bundled with many of the Java EE application servers. The biggest gains to productivity with JSF are the wizards available in most IDEs that generate most of the boilerplate code and configuration for you.

Reason: The quick prototyping of JSF relies on the tooling around it. Maven and Netbeans provide archetypes and wizards to assist with getting started.

STRUTS

Many devs see Struts as a legacy technology, so don't expect fancy code generation in the place of boilerplate code. You need to configure a lot to start prototyping. An example project can be a good starting point. Something on the bright side: Struts has a Convention plugin that enforces some convention over configuration and provides annotations to configure URL mappings and some other stuff. This should speed up things a bit.

Reason: Lots of boilerplate code, no built-in code generation, no external powerful tools.



5. Framework Complexity

Here, we'll discuss how many moving parts exist in each framework and how the complexity of the framework affects you. Do you really want to learn 10 technologies to use a framework? There are also other considerations when choosing frameworks, such as whether the extra features and benefits outweigh the extra complexity levied against frameworks for your application. Remember the old adage, "what you choose in development, you support in production!"

SPRING MVC

Spring is the Mount Everest, the Pacific Ocean, and the Stegodon of frameworks. The base spring framework gives a solid foundation for over 30 subprojects... 30. These mature projects range from your "basic" fully functional

MVC to Spring .NET and spring for Android. Spring MVC architecture is relatively simple, but still there are many layers and abstractions that are sometimes hard to debug if something goes wrong. And it is highly dependent on Spring Core.

Reason: Spring scares away newbies and people who love simple and lightweight things. It is old and mature framework that has numerous amounts of ways to extend and configure it - and this actually makes it fairly complex.

GRAILS

Grails' MVC functionality is covered by Spring MVC; GORM (Grails' Object Relational Mapping) is actually a facade for Hibernate. Everything is glued together with core spring. All of these frameworks are mature, but heavy-weight and Grails adds another layer of abstraction on top of it. Grails also tries to be a full-stack framework by having its own console, build tool and a lot of plugins. All this can make things very complicated when it comes to debugging some nasty issue.

Reason: Grails has all of the complexity associated with spring and Hibernate, and adds another layer of abstraction to that.

JSF

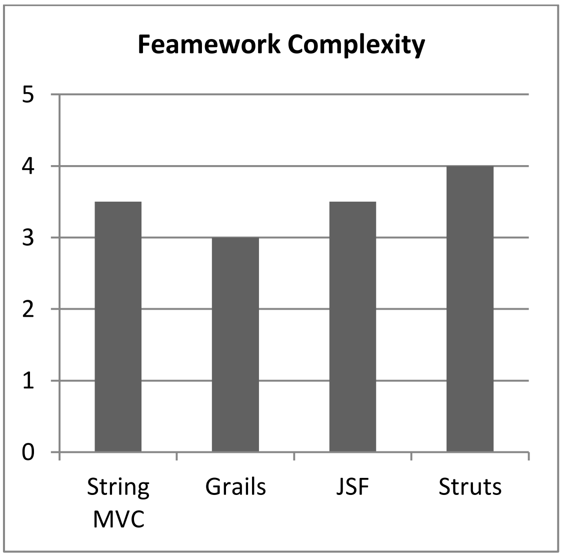
JSF is incredibly complex, and that is the largest downfall of the framework.
However, this is not JSF's fault; this is due to the Java EE specification and the inflexibility of runtimes available. There are open source implementations of the JSF specification that do allow the use of non-Java EE containers like Tomcat, which cuts down tremendously on the complexity of having to run a full Java Enterprise application server (there are middle grounds though, like Web Sphere's Liberty Profile). The complexity does come with the benefit of access to the rest of the Java EE stack.

Reason: When using the Java EE implementation of JSF, we must account for the complexity of the Java EE specification and running a full Java EE server, and one would be hard pressed to claim that the Java EE specification is not complex.

STRUTS

A strut isn't lightweight, but it isn't overly complex either. When a user request comes into Struts, there is an Action (Struts' term for a controller) to be performed and interceptors to be invoked before and after the action. Interceptors can manage logging, security, double-submit guarding etc. Official documentation states that "The default Interceptor stack is designed to serve the needs of most applications. Most applications will not need to add Interceptors or change the Interceptor stack." The result of the action is rendered using chosen view technology. That's the whole magic.

Reason: Struts is pure MVC with a more or less straightforward architecture. No extra components that may add complexity.



6. Ease to Use

How easy it is to pick up a framework for the first time and play with it, learn and get results. Think of it this way: if you gave one of these frameworks to a colleague, how many questions would you have to answer for them? The quality of documentation is always useful, but if you have to refer to the docs, have you already lost the battle?

SPRING MVC

Spring is extensive to say the least and, as a result of that, is not particularly easy to just pick up one day and use. In order to take advantage, you need to know how spring as a whole works, which is its biggest caveat. There are a lot of online tutorials and documentation pages that you could strip down and use as a base, but, overall, spring is for building serious applications with solid foundations, rich user interfaces and a Restful API, which makes it completely unnecessary for anything significantly simpler than that.

Reason: spring knowledge needed. Plain JSP & Controllers do not provide usable, out-of -the-box components and widgets.

GRAILS

Grails is designed to be a rapid development framework and rapid is a direct consequence of ease of use. It is advocating convention over configuration and doing it right. Extensibility is very simple when using plugins (there is a lot of them). One command in the console - and all the dependencies and configurations are managed for you. One downside is that you need to get familiar with Groovy when you learn how to use Grails. But it shouldn't be a big problem since Groovy is so Java-like anyway.

Reason: Designed to be easy to use without any complex configuration, and Groovy should be more or less easy for Java devs.

JSF

JSF tries to provide an easy to use framework for creating reusable web components in your Java applications without a large learning curve. JSF 2.2 was recently ratified and released as part of the Java EE 7 specification. JSF is very easy to get up and running and often does not require extra downloads or configuration as the necessary code is bundled in any Java EE compliant application server. Assuming it's not enabled already, getting JSF support in your application server can be as simple as enabling a checkbox. There are several great tutorials for JSF, some written by Oracle and others by third parties (MyFaces and others).

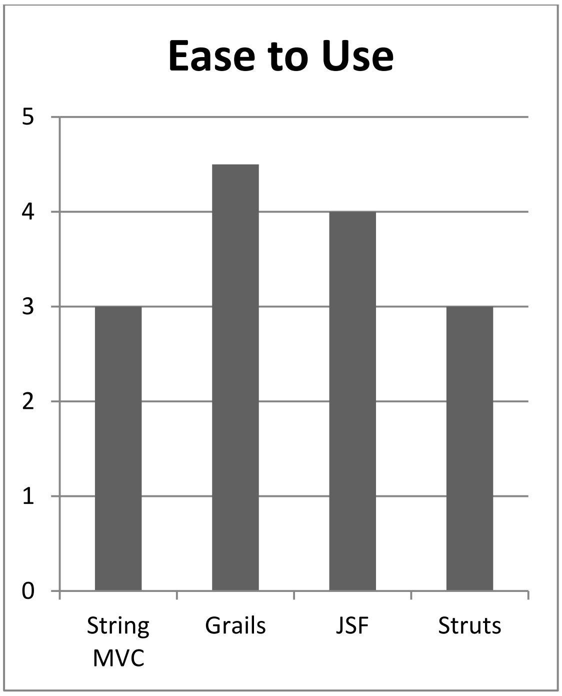
JSF can be a little confusing at times however, Java EE can be overwhelming and incredibly useful features can seem obtuse or obfuscated by poor documentation or easy to miss in the corpus that is the body of Java EE features. Documentation for JSF is often incredibly specific to certain tailored environments and straying from the happy path often leads to frustration and woe for hopeful and aspiring Java EE developers.

Reason: JSF's tooling makes it easy to use and there are no external dependencies as long as you stay within the Java EE ecosystem, which JSF leverages well. There are also several built in components that provide tremendous benefit and some third party component libraries that look great.

STRUTS

A strut is definitely not the easiest to use. Let's say you need to implement a new business spec with simple CRUD operation over an entity. You need to create actions with probably different action results, URL mapping/page navigation configurations in XML, form beans classes with their validation configurations in XML and finally, the view templates. Good luck with all that. Don't forget that you need to make sure that the entire names match in classes/ configuration files / views. Compiler won't do that for you.

Reason: Lots of classes/ configuration files need to be written for a single component.



7. Throughput/Scalability

What you choose in development will need to be supported in production. Throughput and scaling are very important factors as you don't want to bottleneck your application because of a framework decision. This category takes a look at what each framework has to offer to help you application when there are more users and requests than lines of code!

SPRING MVC

Spring applications are meant to scale as spring is used in large-scale applications worldwide. It contains necessary components for parallel processing and solutions like EhCache can be attached easily to scale memory. Spring Batch enables to build multi-threaded apps, partition applications and do bulk processing. Spring apps can be modularized and different modules can be set up on different hosts. These components can talk with each other using JMS (Java Message Service) for example.

Reason: There is wide variety of options how to scale apps using in-memory caching or dividing applications into different parts and connecting them with messaging.

GRAILS

Grails is an abstraction over spring and Hibernate. Are those two scalable? Yes. Will this extra layer negatively impact the throughput? Unlikely, it's all Java byte code in the end. Still not sure? Read the Testimonials and Sites sections on the Grails' website to see what kind of big and fast applications are build using Grails.

Reason: Standing on the shoulders of giants - spring and Hibernate.

JSF

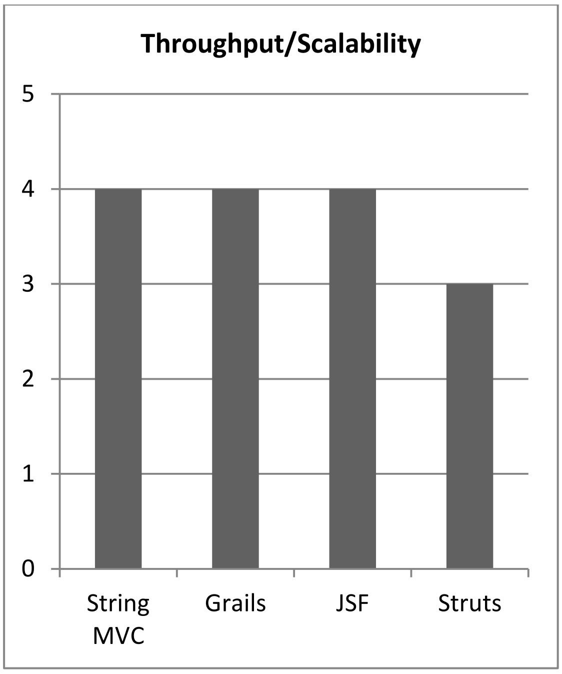
JSF applications can be performing, but really the performance gains come from clustering Java EE application servers. JSF itself does not provide explicit support for asynchronous calls, and relies on the Java-EE specification to provide @Asynchronous and @Schedule annotations on the business logic EJBs.

Reason: JSF provides annotations that are useful for job management with Asynchronous calls.

STRUTS

Similarly to Wicket, Struts is a fully server-side framework. If you need more power for rendering the pages, then adding more servers to your cluster should be the answer. Luckily, Struts-jQuery AJAX plugin can make things a bit more asynchronous.

Reason: No extra features that would improve scalability, other than AJAX support.



8. UX, Look and Feel

One of the most important features! What does the end user see, what is their experience of the application and UI? It's pointless having the most amazing framework if the user experience is poor. The scores in this section also reflect how hard it is to create the UI with each framework.

SPRING MVC

Spring MVC has a very rich feature set to develop and maintain code on the server side, but despite its extensity it still doesn't provide any rich framework for building awesome interfaces. No matter, you can still use it to write a solid backend and use another framework intended for building your UI. It's versatile and plays well with others, so, really, the world is your oyster.

Reason: No re-usable and nice components. But templates are easy to manage and create.

GRAILS

Do you need something complicated in the UI? Then either create it yourself in GSP with a possible addition of JavaScript/CSS or find a specific plugin that will do the job for you. Most certainly you won't have to implement a date picker yourself, but be ready to fine-tune your layout CSS by hand. The auto-generated project skeleton has some basic view templates and CSS to start with. Plugins include integrations with jQuery and Twitter Bootstrap, rich UI AJAX components and a bunch of other things.

Reason: Plugins provide integrations with popular JS frameworks and include many rich UI components.

JSF

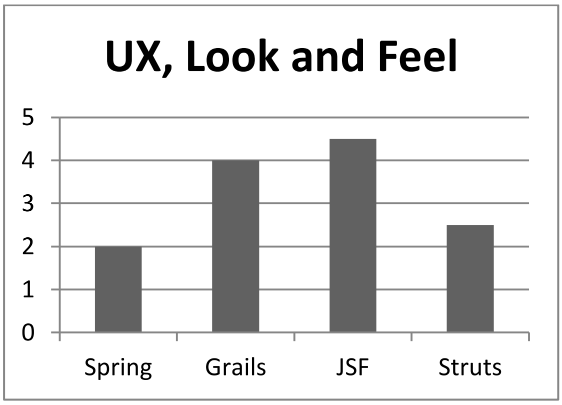
JSF is great for development teams that want to create great full-featured UIs without having to become JavaScript masters. There are a lot of great components already in the catalog, and there are also fantastic additions to JSF to provide even more high quality components. IceFaces and PrimeFaces are two examples of great JSF catalog additions. With the advancement of these extra components, some of the web applications created with JSF is indistinguishable from their desktop counterparts.

Reason: The library of built in components is very full featured and there is also an extensive 3rd-party catalog available.

STRUTS

When you’re using Struts, most of the UX is up to you. Grab a front-end developer to write your HTML, CSS and JS and fill it with your data using JSP, Velocity or Freemarker templates. Struts have 3 predefined themes: simple, xHTML, and css_xHTML. They determine the HTML generated out of the JSP tags. New themes can be created and existing ones can be modified. Struts supports AJAX out of the box, thus there is no need to write JavaScript by hand for it.

Reason: No built-in or 3rd party components. Some built-in AJAX support.



9. Summery

Rapid application prototyping

Here we see quite a range of scores, with Grails and Play coming out on top, due to the mostly to the scaffolding support. Vaadin was close behind and the Vaadin Directory of plugins is worth a mention. Spring and Struts really struggled here, with few out of the box components.

Framework Complexity

Vaadin, GWT and Struts prove to be the simplest of frameworks with fewest dependencies, while Play struggles, particularly with its reliance on Scala knowledge and SBT.

Ease of Use

Grails and Vaadin did well when it came down to the ease of use, particularly due to the configuration, and design mode. Spring and Struts again suffered, due to the amount of knowledge required to use and the amount of configuration required.

Throughput/Scalability

Play was easily the winner here as its features are so suited to scaling and throughput, supporting asynchronous applications with Akka actors. GWT and Vaadin also did well as their model is based around client side execution.

UX, Look and feel

Vaadin and GWT excelled here, with neat usage of customizable themes and the ability to inherit from existing themes. Presentation wise, they are both very easy on the eye. Spring MVC and Struts both suffered here, clearly not one of the goals of the framework.

As with our Great Java App Server debate, not all frameworks are created equal. In the same way that comparing Web Logic with Jetty doesn’t exactly make sense to some readers, some of you will surely find issue with comparing Spring MVC, which is part of a much larger framework ecosystem, to Vaadin or Struts. Also, in reality you might choose a combination of frameworks, rather than just one, making use of the best aspects of each framework. This is why we are preparing a second part to this report, where will measure these same frameworks from the use-case perspective, thereby slightly muting the straight numbers seen in this report. After all, a large team of developers creating big enterprise apps might be prepared to spend more time getting set up with a more heavyweight framework rather than a hobbyist developer.