THE CURIOUS CODER'S JAVA WEB FRAMEWORKS COMPARISON

SPRING MVC, GRAILS, VAADIN, GWT, WICKET, PLAY, STRUTS AND JSF

TABLE OF CONTENTS

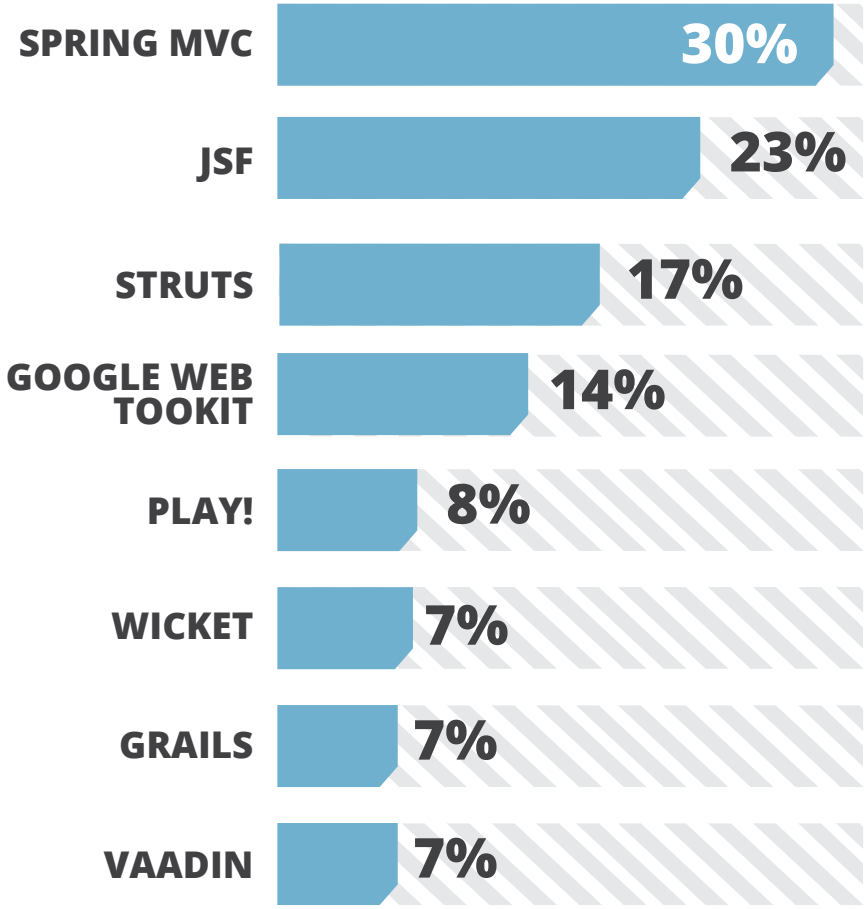
INTRODUCTION
LET'S GET CURIOUS...

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 chose 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.

Why do we need Web Frameworks?

Well, coding good-looking web applications in Java isn't super easy at time. In fact, let's just say it sucks. 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?

We thought it made sense to follow up on the Java Web Frameworks section of our popular Developer Productivity Report and see what we had in there 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
- Documentation & Community
- Framework Ecosystem
- Throughput/Scalability
- Code Maintenance/Updates
- UX, Look and feel

We're going to compare and contrast each Web Framework in this report against each category above scoring and placing each of them based on our findings. The Java Web Frameworks (and versions) we will be discussing in this report are:

Spring MVC 3.2.3, Grails 2.2.2, Vaadin v7.1.1, GWT 2.5.0, Wicket 6.8, Play 2.1.2 , Struts 2.3.15.1 and JSF 2.2.

What can I look forward to in the next report?

In order to avoid a 9000-page report, we wanted to separate it into two parts. In this part, we look at each framework objectively and compare them. In the second report we will take a look at different application types and styles, matching the most appropriate Java Web Frameworks to each, from the information and scores from this report. Each application type will care about the categories mentioned in this report to varying extents, which will aid us in weighting each of the categories for the application types. So save some popcorn, keep an eye out for the trailers and make sure you come back for the sequel to this blockbuster report.

PART I
WARMING UP

It’s really important to be able to launch yourself into a new language or project and be productive quickly. In this first part, we look at how quickly you can begin **prototyping an app**, how **complex** the framework is, the **ease of usability** and what's up with the framework's **documentation & community**.
Let's go!

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 Petclinic 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.

Score: 

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.

Score: 

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

VAADIN

The Vaadin directory is a component repository which provides existing components that users have implemented and contributed for others to use. If you choose to use something in the repository, you simply need to download the JAR and add it to your project and use it in your code straight away - template-tastic. At the time of writing this post, there are 365 components in the directory for you to use, one for every day of the year! You can't get much more rapid than that ;)

Also, the design mode provides a drag and drop mechanism to add components and the code is generated for you, providing you with the exact layout you want, without even having to use the keyboard!

Score: 

Reason:
Scaffolding and the Vaadin directory are both very impressive.

GWT

There are lot of pre-canned widgets for quick use but really anything you can do with JavaScript and the browser's DOM can be done with GWT. If you prefer designing, you can also use GWT's built-in Design Mode, an easy drag and drop interface with automatic code generation.

It's no Grails, but the goal is to allow developers to productively develop web apps without being an expert in JavaScript, XMLHttpRequest, and various browser quirks.

Score: 

Reason:
GWT is easy to create and maintain web apps widgets, and the built-in compiler takes care of efficiency and browser support.

WICKET

With its clear MVC model, clean HTML and a wicked component module, you can create new applications with reusable components quick and easy. The existence of markup files for every component gives you a clear separation of concerns between the controller deciding what to display and the view that specifies how to display. This not only reduces development time but also gives your web designers the confidence to modify the HTML code without worrying about crashing and destroying everything. Win!

Wicket does use a model inheritance for it's components, which isn't great due to a bit steeper learning curve.

Score: 

Reason:
HTML and component structure with no need for configuration files. Separation of components and how they display. Model inheritance for components.

PLAY

Play is super simple to get started. The origination story of the framework essentially boils down to Java developers being envious of Ruby on Rails developers having a super fast prototyping framework. Play has a binary component similar to the Rails component of RoR, which is used for scaffolding and makes Play a little unconventional as a Java web framework. The Getting Started section of the Play framework documentation is a great basic tutorial and there's a very low barrier to entry for developers to become at least moderately productive.

Score: 

Reason:
Play has great documentation and the scaffolding works wonderfully.

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.

Score: 

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

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.

Score: 

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

SCORE SUMMARY - RAPID APPLICATION PROTOTYPING

Framework Complexity

This section is where we explore each framework's construction. 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, 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.

Score: 

Reason:

Spring scares away newbies and people who love simple and lightweight things. It is old and mature framework that has numerous amount 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 it's own console, build tool and a lot of plugins. All this can make things very complicated when it comes to debugging some nasty issue.

Score: 

Reason:

Grails has all of the complexity associated wtih Spring and Hibernate, and adds another layer of abstraction to that.

VAADIN

The Vaadin Framework currently builds upon a flavour of GWT (lightly altered), which is a mature framework. The way you'd go about developing a Vaadin project is similar to GWT in the sense that you could create a hierarchy of components, but learning GWT is not a prerequisite to be competent at using Vaadin.

Score: 

Reason:
Slightly more complex than GWT, as it embeds a version of it, but not enough to drop it any marks.

GWT

GWT is a pretty complex, if small framework for being so easy to use. You only have to worry about the Java code and what components you want to put where (or just drag and drop components using the Design mode and let GWT auto-generate all that code for you) and GWT cross-compiles that code into highly-optimized JavaScript that works across all major browsers.

Score: 

Reason:
Pretty complex framework with an interesting execution that makes your life easy to develop with, either through code or through the drag and drop Design mode.

WICKET

For being an MVC, Wicket can be fairly straightforward. It has clean HTML and a large library of components and a wide range of model objects that the components can use to retrieve or push data. However, the model inheritance can be pretty difficult to grasp initially and, while it increases code reuse, it reduces readability and is unnecessarily complex.

Score: 

Reason:
A straightforward framework but with an unnecessarily complex model inheritance system that is useful but also difficult to grasp.

PLAY

Play is quite a complex framework, and there are a lot of moving pieces. This makes sense given how robust of an ecosystem the Play framework provides. The framework comes with Netty, SBT, Akka, a Scala templating engine, and several other modules built in. Play emphasizes convention over configuration, so the framework is responsible for scaffolding much of the glue and configuration between the modules, which does cut down on the complexity experienced by you, the developer.

Score: 

Reason:
Play provides an entire ecosystem instead of just a framework, which helps provide a lot of features. However, you do need to know something about Scala, which adds additional complexity.

STRUTS

Struts 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.

Score: 

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

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 WebSphere's Liberty Profile). The complexity does come with the benefit of access to the rest of the Java EE stack.

Score using JSF in a 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.

Score using JSF in an OS implementation: 

Reason:
When using an OS implementation of JSF, we need to account for the complexity of the underlying framework that JSF is being run on top of.

SCORE SUMMARY - FRAMEWORK COMPLEXITY

Ease of Use

This section is all about 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 as well as the Petclinic sample application 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.

Score: 

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.

Score: 

Reason:

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

VAADIN

Vaadin takes away the bulk of the GWT boilerplate code away from the developer, so that the code looks a lot neater. A great feature for 'ease of use' is the design mode. The design view allows us to drag and drop components onto a canvas and provide the logic code behind each component. The code which creates and sets up the UI for each component is created for us, magically, which we can see back in the source view. There are also many videos and documentation pages set up for new users to ease the new user learning curve. These really do take you by the hand and get you up and running. If you're not using the design mode, the code can get a little hairy but not too bad. Also visualising your changes in the runtime can often be a bit slow.

Score: 

Reason:

Super easy to use, particularly from the design mode. The Vaadin directory also helps. If you run into problems, you have plenty of support around to help out.

GWT

Whether you're a coder or a designer, GWT has a good, fairly straightforward structure. The component structure for the coders out there are easy to work with and the widget options are numerous. The code is easy to read, modify and reuse, and if you are already familiar with client-side storage that is easy to work with as well, the only problem is, there's so much of it. The code is incredibly similar to JavaScript (it does translate it to JavaScript at compile time after all) so if you'd need to be familiar with it to develop quickly.

GWT Designer, while not for everyone, is a powerful bi-directional Java GUI designer that lets you create user interfaces with tools for intelligent layout assist, drag-and-drop, and automatic code generation.

Score: 

Reason:

Code is easy to read and write and there's a Design mode is there for the non-coders. Client side storage is a little hard to grasp if you haven't used it before.

WICKET

Wicket keeps things simple - it lets Java developers do what Java developers do best: Java. There is a clear separation of markup (HTML/XML) and logic (Java code) which lets your Java devs worry about the Java and your web designers make changes to the markup templates of an application without worrying about breaking everything (and a huge sigh of relief just sounded from the design corner).

Wicket is also designed to do a lot of the heavy lifting for you, so boilerplate and repetitive code is kept to a bare minimum. View Components can be easily generated from metadata such as annotated “bean” classes and there are built-in utilities in extension modules that allow the developer to bind these components to back-ends.

Score: 

Reason:
HTML and Java code focused, no need for silly configuration files. Model inheritance can be a little hard to grasp initially.

PLAY

Play is really simple to get started with, however it’s very complex for a developer to take full advantage. One of the best things about Play is the ability to go from zero knowledge of the framework to leveraging some of the more basic capabilities in less than 10 minutes. Features like scaffolding and native JSON processing are immediately available to even the most eager developer, while features like asynchronous calls through Akka Actors may require a bit more research into the documentation.

There are a few knocks against Play in the Ease of Use department; the most obvious learning curve issue with Play is integrating it with your existing build tools and environment. Play tries to provide an entire ecosystem by itself, which can make it very non-trivial to swap items for your existing pieces of infrastructure and tooling, using SBT instead of Maven or Ant for example. The other potential issues with Play in terms of usability are backwards compatibility with existing code bases built on previous versions of Play and Akka Actors. The Play migration guide is very comprehensive for migrating 2.0.x applications to 2.1.x versions of the framework, however migrating a version 1.x application to 2.x is not for the faint of heart.

Score: 

Reason:
While simple to get started, the two things that really keep Play from being really easy to use are the complete backward incompatibility between versions and the requiring Scala to use templates.

STRUTS

Struts 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 all of the names match in classes/configurational files/views. Compiler won't do that for you.

Score: 

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

JSF

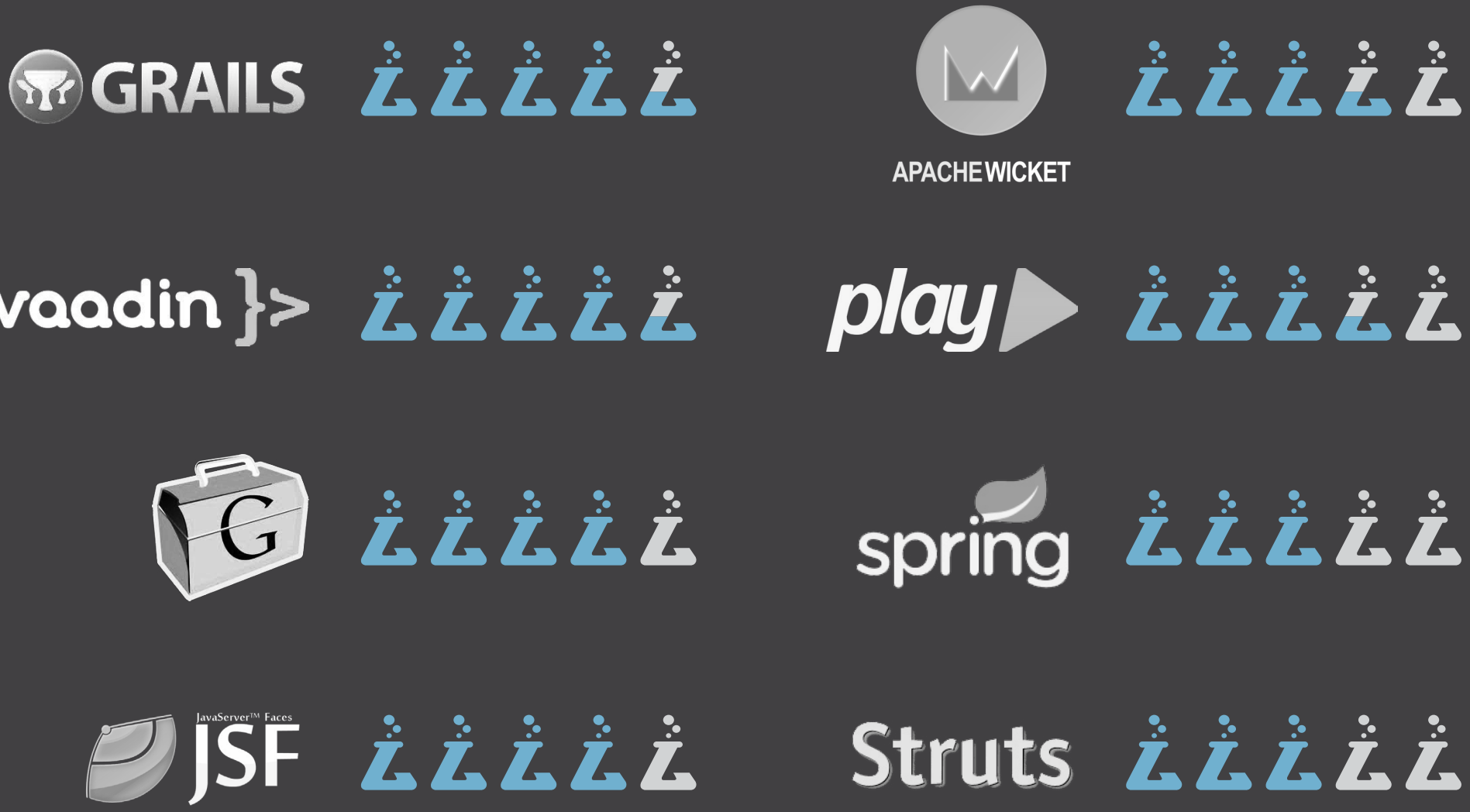
JSF tries to provide an easy to use framework for creating reusable web components in your Java applications without a large learning curve. JSF 2.2 was recently ratified and released as part of the Java EE 7 specification. JSF is very easy to get up and running and often does not require extra downloads 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.

Score: 

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.

SCORE SUMMARY - EASE OF USE



Documentation & Community

For a good framework to continue improving, especially when not financially backed by a larger parent company (ehem, like VMWare, or the confusing new brand Pivotal), it's important to have a couple things going for it. First, excellent **documentation** so that new users can quickly attain guru level easily, and secondly the willpower to not only establish the seeds of a **community** early on, but also to take a large part in interacting and fostering the community. Good examples of this are GitHub and NetBeans)

SPRING MVC

Being the most used framework by Java developers definitely has its perks. There is so much information out there about Spring that a whole report could probably be dedicated to just listing and analyzing the available resources. The Spring website itself provides numerous tutorials both in video and written format, and it's useful for beginners to the most advanced developer. There is a link to a GitHub repository with the Petclinic sample application - the go-to for a Spring MVC tutorial. Spring also hosts a blog on their website with articles written by the public as well as Spring team members.

The community is strong with this one. The Spring team actively encourages forking the Spring project repositories in order to make changes, fix bugs and add new features. They allow public access to their JIRA tracker in order to facilitate this. There are numerous Spring User Groups and Java User Groups that discuss Spring topics. Spring even holds an annual conference called SpringOne/2GX. The Spring hosted forums and SO are great places for asking and answering questions and the website blog and monthly newsletter keep developers updated on new technologies and accomplishments.

Score: 

Reason:

Just look at these docs! You can get information about anything and these docs are mostly up to date. But if it comes to starting from scratch as a newbie (finding how-tos and tutorials), then the life will be harder.

GRAILS

There is a huge amount of information out there. Documentation section is actually a wiki, which can be modified by any logged in user http://grails.org/edit/Documentation. It has an official manual, tutorials, screencasts, a sample app and more. If that is not enough, then look through 42k topics in the mailing list or search through nearly 12k grails-tagged questions on SO.

Score: 

Reason:

Awesome documentation and a lot of extras, like screencasts. Big and passionate community.

VAADIN

Documentation? How about a book that you can use to swat an elephant! The Book of Vaadin is a must have reference manual for all your queries. It does cost to buy the book online, but you can download the PDF for free, or pick up the book at one of the many conferences the Vaadin folks sponsor. Online, there is also a bunch of documentation, including tutorials and demo videos, which are extremely easy to learn from and replicate. The community is strong with an active forum, blog, webinars, conference appearances and of course the community contributions on the Vaadin directory. StackOverflow also has many Vaadin questions, but it looks like for a quick response you're better off heading straight for the Vaadin forums.

Score: 

Reason:

Top class documentation, both online and in The Book of Vaadin. Video tutorials and demos are also impressive. Vibrant community.

GWT

GWT has surprisingly extensive official documentation. The majority of the new (beta) GWT project website is tutorials and framework documentation detailing everything from coding basics to using complex UI widgets and handlers. There are numerous FAQs, reference guides and overviews about pairing GWT with other frameworks. The site also has sections for books (dating back to 2006) written about GWT itself and paired with other frameworks (namely Vaadin), case study videos, sample projects, and presentations given at various conferences. Both the sample projects and presentations are available for download from the website.

Google also established a set of discussion groups and mailing lists that help users obtain support for technical issues and allow them to share their latest accomplishments and projects with the GWT developer community. Developers can benefit GWT by getting involved in general discussions on the GWT group and by becoming a GWT Contributor. As a Contributor, developers can share source code patches and updates to the project itself, expanding GWT's features and functionality.

Score: 

Reason:

Great up-to-date documentation, official and non, and active community.

WICKET

Wicket has reasonable documentation and a somewhat active community. The Wicket website, http://wicket.apache.org/, hosts a large compilation of available examples, sample projects, and resource links. The examples provided range from your basic “Hello, World!” to examples detailing specific Wicket framework features, such as fragments and AJAX. On the site there is a Wiki, Reference Library, and a list of books about Wicket. The resources are useful … and a little bit old.

The community is as active as they need to be, if not as extensive as other framework communities. The website lists regular Wicket bloggers with links to their websites - the owners seem to post every couple months about a variety of topics, but the list seems to be a little out of date with only a third of the listed bloggers having updated in the past three months. There is a more extensive list of bloggers on the Wiki. Beyond the blogs, Wicket set up an IRC for more direct support, there are a few dozen companies that provide commercial support, and there are a splattering of community meetups worldwide.

Score: 

Reason:
The documentation is good and the community used to be very active, but recently it’s been slow. The documentation hasn’t been updated since last year.

PLAY

Play has a reasonably sized community with great documentation and the support of TypeSafe. The community as a grass roots component as well with lots of great tutorials on the web, there are even some great and freely available screencasts introducing Play. The Play framework site has full reference level documentation available free of charge and even covers secondary documentation that covers the non-framework pieces of Play, like Akka and SBT.

The community is very active and there are several companies that base their main sites and applications on the Play framework. One of these companies is LinkedIn, and they independently publish documentation for the Play framework community. Bugs reports and feature requests are solicited from the community on a regular basis, which leads Play to have a robust and active community.

Score: 

Reason:
Lots of great documentation from TypeSafe (Play) and external sources. Play’s full API is well documented and there are great tutorials that show off the scaffolding features of Play.

STRUTS

Struts documentation isn't very organised. There are some guides, tutorials and cookbooks both on the official site and 3rd-party websites. After spending more time than you'd like searching, you will actually find a lot of resources: tutorials for beginners, guides on how to integrate Struts with Spring/Hibernate/%LIB_NAME% and detailed examples on how to use different Struts features. The community is big enough, considering the age and popularity of the framework, so you won't be let alone with your problem. But the amount of dead links (e.g. to the "Wiki pages") on the official website makes us a bit worried.

Score: 

Reason:
Awful official documentation. User-written tutorials are slightly better.

JSF

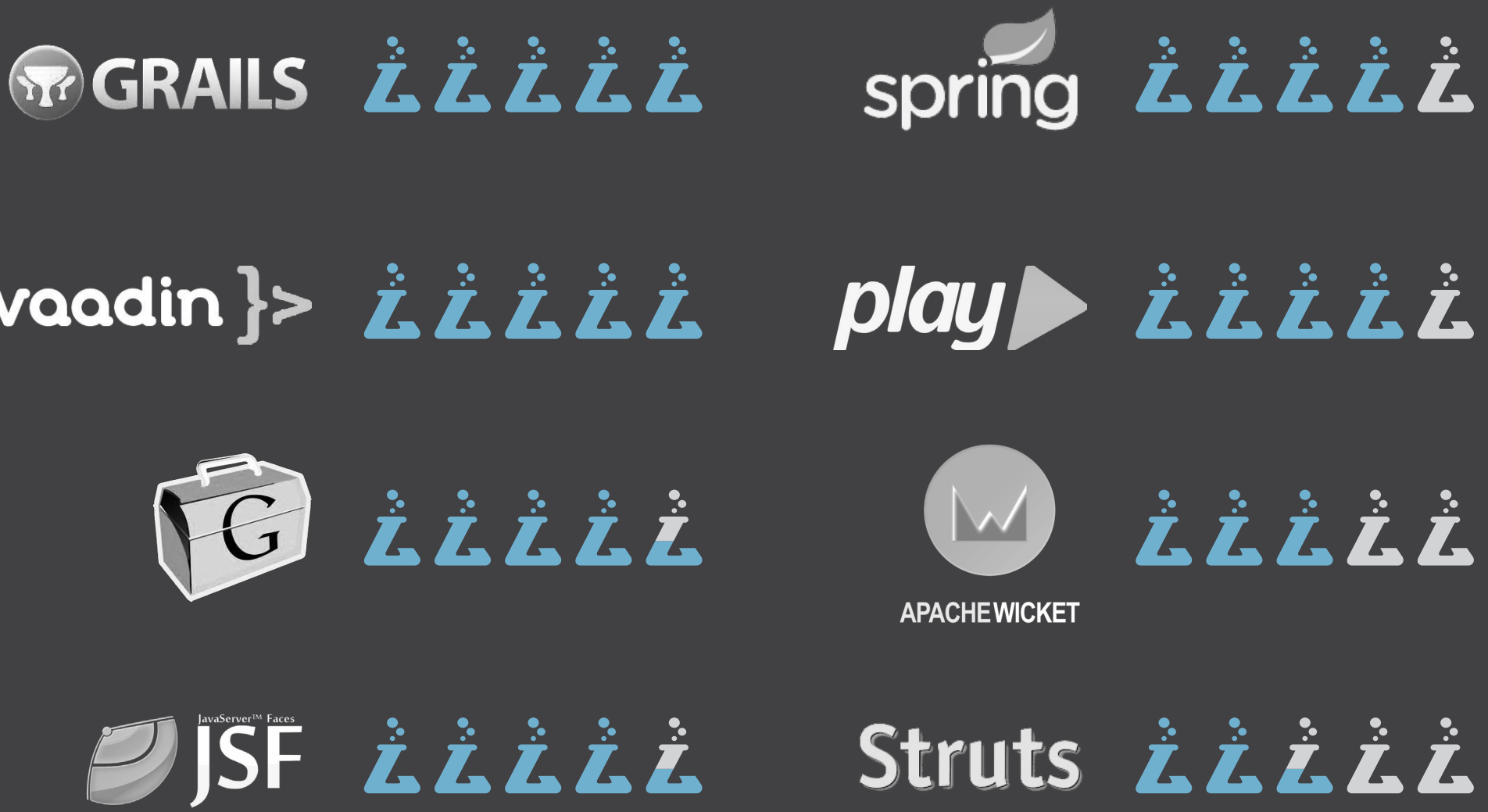
JSF is unlike any other framework in our report for one main reason: it is fully supported and has a reference implementation from Oracle. Unlike the other frameworks in our report, JSF has a top down community with Oracle at the head paying employees to write documentation and create samples and examples. While the Java EE specification is determined by the Java Community Process (JCP), Oracle has a large part in determining and supporting features present in the specification. There is still a community based on JSF and there are a few exciting alternative technologies built on JSF, namely the MyFaces, PrimeFaces, and IceFaces.

Unfortunately, much of Oracle's provided documentation for JSF is built around Oracle's reference implementation of the Java EE tooling stack. Unless you're a NetBeans and GlassFish user, you'll be stuck looking for 3rd-party documentation sources. This is not necessarily a bad thing, there is a lot of great documentation available for JSF outside of Oracle's sphere of influence, however it may not be as comprehensive and cover all of the features you want from the Oracle documentation. A simple search for *JSF 2 tutorial* on your search engine of choice will get you some great links, also youtube and other video sites have decent content.

Score: 

Reason:
JSF has fantastic documentation provided by Oracle and there are tons of great books and online content from third party sources. The only downside of the documentation from Oracle is the reliance of their IDE and Application Server. Requiring Netbeans and Glassfish is definitely a detriment to the documentation and tutorials.

SCORE SUMMARY - DOCUMENTATION & COMMUNITY



PART II
GOING NINJA

It’s time to Go Ninja and look more at the results of what your Web Framework will leave you with in your runtime. You'll want to know how good of a **framework ecosystem** of tools there are available, what **throughput & scalability** you can expect, what it's like to **maintain & update** your code, and how the **user experience** is generally.

Famework Ecosystem

The framework ecosystem covers how extensive the framework is in itself, it's functionality, it's build infrastructure and it's integration with others. Very often, an large ecosystem can mean increased complexity, but not always. This section also depends heavily on user needs, which we will discuss further in the next report.

SPRING MVC

Spring MVC's ecosystem is well-developed. As it is based and dependant on the Spring Framework itself, it can benefit from tools like Spring Roo and Spring Tool Suite IDE. There are no problems with Maven dependencies as everything is available in a public Maven repository and also from SpringSource's own Maven repository. There are 3rd-party solutions like MyEclipse for Spring, which includes scaffolding capability for Spring MVC.

Score: 

Reason:

Still some very useful parts of the ecosystem are commercially available and this makes it not available for everyon

GRAILS

Grails is a full-stack web framework, not just MVC. It contains a lot of stuff out of the box, but luckily, it doesn't enforce it. Don't like the built-in build tool? No problem, you can integrate with Maven or Ant. You've started using some NoSQL solution like MongoDB or Neo4j? You can remove Hibernate from underneath the GORM and exchange it with MongoDB/Neo4j. You want to add caching or security capabilities? There is a plugin for that! Actually a lot of them (OVER 900: http://grails.org/plugins/) that provide a Groovy API for a lot of well-known Java libraries. And they are super easy to install – just add <code>compile ":lib_name:lib_version"<code> to your build script.

Score: 

Reason:

Grails is a full-stack framework, but many provided parts can be replaced, and there is a big library of plugins

VAADIN

Vaadin doesn't really provide an ecosystem as such. But does allow for it within the Vaadin Directory. For example, let's say you wanted to use Hibernate with Vaadin, you'd need to download the plugin, and you're ready to go. There is integration advice and documentation around how you can integrate with Spring, Flex, Liferay, Maven, but not much else. Ultimately, because the ecosystem doesn't exist out the box, when you need to expand your application your complexity will grow as you support the different technologies and their integration.

Score: 

Reason:

Can only go so far with integration, but is not an ecosystem out-of-the-box by any means.

GWT

GWT allows you to quickly build and maintain complex and highly-performant JavaScript front-end apps in Java. The GWT SDK allows you to write your AJAX apps in Java and then cross-compiles the code into JavaScript that works across all major browsers. The SDK supplies a core set of Java APIs, libraries, app server and the compiler, and you can also mix in your own JS code and existing JS libraries.

Score: 

Reason:

GWT SDK with a core set of Java APIs, libraries, app server and compiler. Works well alone but better with others, and is not an ecosystem out-of-the-box.

WICKET

Wicket is a full-featured framework that allows you to create reusable components and build up from those components in your projects to come. It is a server-side framework that has great AJAX support built into it and makes building pages with AJAX components a breeze. If you are a JavaScript fan, you can use your favorite client-side JS library to do things and then have Wicket server the data in any required format.

Score: 

Reason:

Wickets gives you a full MVC with AJAX support, but plays well with others rather than standing alone.

PLAY

Play really shines in this category, as it is bundled with SBT (Simple Build Tool) for building, Netty for a non-blocking networking I/O, Scala is used as a templating engine, and Akka is included for asynchronous operations. Play, unlike many Java web frameworks, includes a binary that is used for scaffolding of your applications. The Play framework is an all-encompassing ecosystem solely developed to increase developer productivity while minimizing the dumbing down of capabilities or shortening of feature lists.

Score: 

Reason:

Play provides everything you need to develop, run, and test your applications. The only real downside is Play's reliance on Scala for templating and SBT.

STRUTS

Struts doesn't enforce anything else than just itself. You may choose from a variety of template engines, build tools, ORM implementations, dependency injection frameworks. There are some plugins (https://cwiki.apache.org/confluence/display/S2PLUGINS/Home) that can help you with integrating Struts with other technologies. If there is no plugin, then most likely there is a user-written tutorial out there somewhere.

Score: 

Reason:

Another pure MVC that is ready to be integrated with other technologies.

JSF

JSF's ecosystem includes component libraries such as Richfaces, OmniFaces, Primefaces, IceFaces, ADF Faces as well as the extension points, including PrettyFaces, GMaps4JSF and Mashups4JSF. The real value however is being part of the Java EE specification. JSF can be easily added to your existing Java EE environments and immediately receives all of the benefits inherent in the Java EE ecosystem. JSF can be leveraged on non-EE containers as well, with open source support from the Apache Foundation with their MyFaces implementation of the JSF specification.

Score: 

Reason:

Because it is part of the Java EE specification, JSF receives all of the benefits and tooling of Java EE. That includes great IDE support and a lot of functionality built into the application servers

SCORE SUMMARY - FRAMEWORK ECOSYSTEM

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.

Score: 

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 bytecode 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.

Score: 

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

VAADIN

As Vaadin relies so heavily on it's GWT implementation, it leverages many of it's positives (while trying to reduce the pitfalls). One of the advantages is the way in which GWT turns Java code into JavaScript. Why is this an advantage? Well, JavaScript runs in the browser and so as an environment scales, the server back ends don't run all the logic, the clients take a large portion of it. Does Vaadin add any complexity to the scaling nature of GWT? We don't believe so.

Score: 

Reason:
Takes the scaling value from GWT and doesn't add any additional scaling complexity.

GWT

If GWT was created for one thing, it was created for scalability. GWT code looks similar to many other standard toolkit APIs but with asynchronous calls (though a Remote Procedure Call infrastructure is provided to make it feel like standard event processing). The included compiler is what makes GWT stand out. The Java to JavaScript compiler hides browser differences so you can focus on the actual code without worrying about adapting to different browsers and browser versions; it analyzes your source for unreachable code and removes it; and it eliminates method call overhead by inlining code. So, whether you have a small app or a big app, GWT will analyze and compress the returned JavaScript to be as efficient as possible no matter what browser you're using.

Score: 

Reason:

GWT's included compiler that converts Java to JavaScript is magic for efficiency and scalability.

WICKET

Wicket comes up a little short on the scalability front. Because it is a server-side framework, building the resulting page to be rendered to the user is done on the server and will thus consume server resources. It may not be the best option for applications where throughput is a top priority. However, it does have great AJAX support so Wicket could be used to render the initial page skeleton and have the rest of the communication with the server done via AJAX requests. Just make sure you configure your components to use AJAX if possible.

Score: 

Reason:

Wicket works well for scalability if that is your goal when developing the foundation; otherwise, you're better off using another framework that doesn't have such a huge server resource consumption problem.

PLAY

Play is incredibly scalable with high throughput. When coupled with the native integration for Akka actors, Play enables ridiculously performant asynchronous applications ready your Big Data needs. The notion of Future really enables Play to deliver on the promise of high concurrency and large datasets. Built in JSON support assists with integrating into other web services, enabling your code to be more modularized. Even Netty is non-blocking, which should cut down on useless waiting.

Score: 

Reason:

Play uses Akka actors to be incredibly scalable and allow for high throughput. It does require learning how to use Akka.

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.

Score: 

Reason:

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

JSF

JSF applications can be performant, 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.

Score: 

Reason:

JSF provides annotations that are useful for job management with asynchronous calls.

SCORE SUMMARY - THROUGHPUT/SCALABILITY

Code Maintenance/Updates

Releasing you first version of code is the start of a wonderful journey of support, migration and new features. But how wonderful will that journey actually be? Will you be in a state where each project, using the same framework is configured differently? Planning for the future saves regretting for the past.

SPRING MVC

Unlike the convention-over-configuration frameworks like Grails and Play, Spring MVC is heavy and dense despite taking advantage of the Model-View-Controller paradigm (Spring -> MVC <-). Updating and code maintenance is manageable if you're already familiar with the intricacies of the framework and the project itself, but if you're just diving in it can be a little overwhelming and slow going. But still, annotations enable to access all kinds of data in a declarative manner and this way, not much hacking is needed.

Score: 

Reason:

Annotations are cool. They make life so much easier. Multiple ways to configure. Server side logic is easy to maintain but if comes to managing layouts and complex UI-s, it will be hard.

GRAILS

Just like the Play framework, Grails follows the convention-over-configuration principle, which makes maintenance easier. It enforces the project structure, thus most Grails projects look very similar. Groovy is less verbose than Java, thus less code need to be written. Hot code reloading makes the updates even more fast.

Score: 

Reason:

Convention over configuration makes most of Grails projects look similar and if you are new to Grails - those conventions are easy to grasp.

VAADIN

So you want to change your Vaadin implemented UI but don't want to change all the code in the world? Well, it's not actually that bad, as many of the Vaadin components share many of the same characteristics, so if you wanted to switch from one component to another to capture input slightly differently, e.g. from a drop down list to a table of elements, it is surprisingly easy to do so. Very often it's just a case of switching the class names around and maybe touch up one or two attributes and you're good to go. Code updates can be quite slow, but JRebel integration is supported.

Score: 

Reason:

Very good code reuse and component class hierarchy model, as well as switchable themes, shame it can be a little slow to update.

GWT

Being a component driven framework with a JavaScript-esk feel (it does compile to JavaScript after all) gives GWT a leg up on manageability and extendability. The code is easy to step through line by line and the availability and use of widgets allows a lot of code reuse. The automatic code generation through the Design mode is relatively organized and manageable as well.

Score: 

Reason:

Very easy to maintain and update but automatically generated code from the Design mode can be hard to deal with.

WICKET

Maintaining and updating code written for the Wicket framework is manageable and further helped by its execution of the convention over configuration methodology. The user components are based on inheritance and so are easy to debug, and the separation of style and logic means you can reuse a lot of your code and easily change the look of your web application without worrying about breaking the logic. HTML templates are easy to modify and errors come out early and clearly.

Score: 

Reason:

Separation between HTML and Java allows for easy design changes without ruining existing logic. No additional configuration files to deal with.

PLAY

Code written for the Play framework is readable, as the framework follows a convention over configuration methodology that makes most Play framework projects look very similar in structure. This enables a developer to switch between applications without having to relearn the ecosystem for every project. The built in templating system also helps with the readability of code and can help create incredibly powerful and useful applications with lower Lines of Code counts. Play also has reloading of components and code built in which definitely helps cut down on time during the development phase of your applications.

Score: 

Reason:

The convention over configuration aspect of Play leads to readable code, however the reliance on Scala does limit the readability to Java-specific developers.

STRUTS

Similarly to Spring MVC, maintenance should not be a big problem if you are familiar with Struts, but if you are new to the project, the learning curve can be steep. Even if you have experience with Struts, your new project may have a bit different structure and different configuration.

Score: 

Reason:

Lots of different aspects to grasp. Two Struts projects can also look pretty different.

JSF

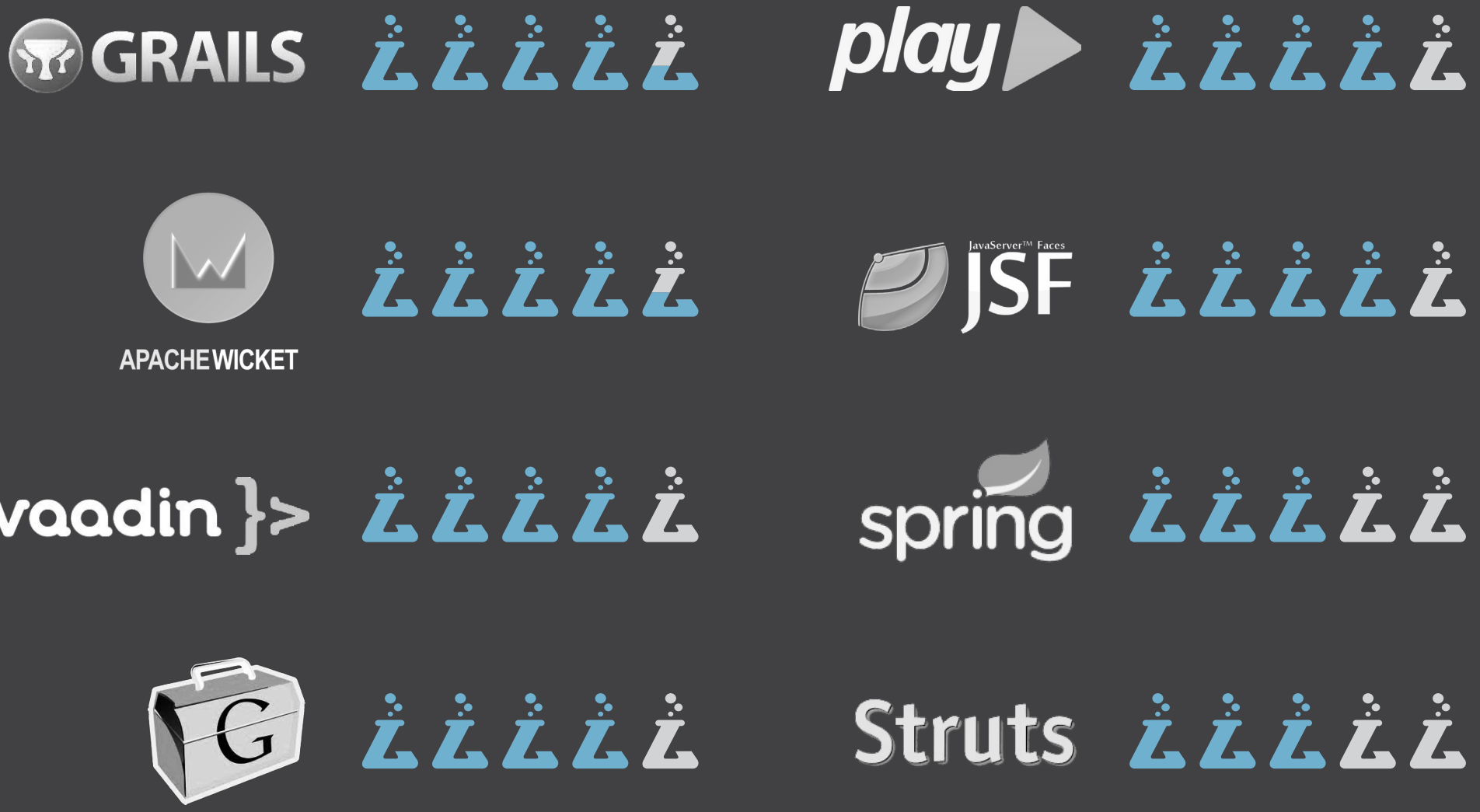
JSF projects leverage existing design paradigms such as Model-View-Controller to help enforce maintainability and readability. The framework itself does not have any features that particularly impact readability or maintainability. One of the stated goals of JSF is to help developers produce better code, but we do not see how they facilitate that.

Score: 

Reason:

JSF leverages pure Java syntax in well-defined paradigms to create projects that any standard Java EE developer can step in and work on.

SCORE SUMMARY - CODE MAINTENANCE/UPDATES



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 extensivity 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.

Score: 

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.

Score: 

Reason:

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

VAADIN

Vaadin themes are collections of layouts for components, custom images and CSS, The whole idea to have different themes is to separate the logic of components from their look and feel and being able to switch themes without changing the logic. A theme follows a specific directory layout, and with them you can specify the css for the default Vaadin components or create custom components.

One of the coolest things about themes is that you can inherit from existing themes, so when you need just to tweak a bit, you don't need to create a monster or hack themes installed by default. This means you can inherit from a theme and just add the missing pieces.

Score: 

Reason:
This is what Vaadin was made to do! Its themes are a joy to use, and the results are very slick and pleasing to the eye.

GWT

GWT is very versatile when it comes to customizing and awesome-izing the look of your web apps. Because it's a component-based framework, you can modify the look of your app at any time without breaking the logic. This separation also helps the application render more quickly, consume less memory and make it easier to tweak during edit/debug cycles.

GWT allows you to associate style sheets with your project in order to further customize it. The GWT theme style sheets are associated with the application as a whole - there are three options: Standard, Chrome, and Dark. You can also create an application stylesheet(s) where you can define specific styles for your application in CSS.

Score: 

Reason:
Theme hierarchy allows you to modify individual component look without break overarching application theme which allows for a great deal of easy and pretty customization.

WICKET

Like the other component based frameworks, Wicket gives you a large selection of out-of-the-box widgets to choose from. If none are to your liking, the website links to a handful of good websites where you can find more Wicket Components, HTML, CSS and JavaScript to further customize your web application. It's a very versatile framework and plays well with others, so the UX choices really fall to you!

Score: 

Reason:

UX is pretty mediocre out of the box (solid 90s look) even with its large library of pre-canned widgets, but paired with a good framework your apps can look awesome.

PLAY

Play has some basic themes, however there is not widespread support for component libraries built in. Play applications usually leverage LESS CSS and CoffeeScript mixed with templates instead of the CSS and JavaScript leveraged by traditional Java web application

Score: 

Reason:

Nothing mind-blowing, but Play has some enhancements over standard JavaScript, CSS, and XHTML by providing LESS CSS and CoffeeScript mixed with Scala templates.

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 has 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.

Score: 

Reason:

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

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 are indistinguishable from their desktop counterparts.

Score: 

Reason:

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

SCORE SUMMARY - UX, LOOK AND FEEL

PART III
THE RESULTS!

Now we come to the fun part! Let's collate all our results and make sweeping statements about which Web Framework you should all be using, mocking those who are not using what we consider to be their best option... Kind of...

The Results

First of all, lets recap all the scores in each category for all frameworks. We've marked the (joint) winners for each category in blue:

	Rapid Application Development	Framework Complexity	Ease of Use	Documentation & Community	Framework Ecosystem	Throughput/ Scalability	Code	UX/Look and Feel
Spring MVC	2.5	3.5	3	4	4	4	3	2
Grails	**5**	3	**4.5**	**5**	**4.5**	4	**4.5**	4
Vaadin	4.5	**4**	**4.5**	**5**	3	4.5	4	**5**
GWT	4	**4**	4	4.5	3	4.5	4	**5**
Wicket	3.5	2.5	3.5	3	3	3	**4.5**	3.5
Play	**5**	2	3.5	4	**4.5**	**5**	4	3
Struts	2	**4**	3	2.5	3	3	3	2.5
JSF	3	3.5	4	4.5	4	4	4	4.5

Interestingly each of the frameworks we looked at either wins, or jointly wins at least one category, apart from Spring MVC and JSF. Let's see how this feeds into the grand totals:

Grails	**34.5**
Vaadin	**34.5**
GWT	33
JSF	31.5
Play	31
Wicket	26.5
Spring MVC	26
Struts	23

Well, there's certainly no runaway winner, in fact, the top 5 are only separated by 3.5 points! But then a gap separates them from Wicket, Spring MVC and Struts. **Vaadin** and **Grails** just take the top spot, which seems fair, looking at how they stood up to our tests.

Grails came out on top for 5 of the 8 categories, doing particularly well around the Rapid application development, documentation and community with top marks. The only area it struggled a bit with was the framework complexity category, mostly because of it's reliance with Spring, GORM, Groovy and more adding layers and layers of abstraction.

Vaadin won 3 of the categories, and also got a 3/5 in one of the categories, but was typically matching Grails blow for blow through the competition. It excelled in its documentation and community category which is excellent as well as the UX, look and feel category, which to be fair is what Vaadin is all about.

GWT wasn't far behind, which you'd expect given how similar it is in functionality to Vaadin. In fact it matched it in all categories apart from 3, where Vaadin only pulled ahead by a half point in Rapid App Development, Ease of Use and Documentation and Community: arguably some good reasons that Vaadin was maybe created over GWT.

Interestingly, after mentioning that **JSF** hadn't won any of the categories, it sits in 4th place, mostly due to it's solid performance throughout, only dropping below a 4/5 twice, in Rapid Application Development and Framework Complexity. It's worth mentioning that calling JSF a "web framework" is often debated due to it's inclusion in the Java EE stack, so we're glad to see it perform strongly considering it's specific place in Java.

Play could have finished higher, but for the framework complexities and ease of use. It really isn't a framework that is very easy to get up to grips with quickly. We use Play at ZeroTurnaround and this is certainly the vibe from the engineering team. Once you get used to Play you can create applications very quickly, but where it really excels is in the scalability and throughput.

Spring MVC, Wicket and **Struts** didn't do particularly well up against the other frameworks. Spring did reasonably well in quite a few categories, but really fell short in the UX, Look and Feel category as well as the Rapid Application Development category. Struts was poor across the board as was Wicket, apart from one victory in the Code Maintenance/Updates category which really put it above Spring in the rankings.

IN THE END
GRAILS & VAADIN
ARE THE BIG WINNERS,
FOLLOWED REASONABLY CLOSELY BY GWT, JSF, AND PLAY.

THERE YOU HAVE IT. NOW YOU CAN GO HOME ;-)

SUMMARY OF FINDINGS
AND GOODBYE COMIC

Wow, you made it to the end! Or maybe you skipped to the end ;) Either way here's the report summary in manageable byte-sized pieces.

Summary

For those of you pulling a TL;DR (too long, didn't read), here is a sum up of each category we measured.

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.

Documentation & Community
Grails and Vaadin again took top spot with their vibrant communities and extensive docs. Struts and Wicket have problems with their official documentation, so fared badly again.

Framework Ecosystem
Grails and Play both have good ecosystems, including build support all the way to tools and plugins. Spring MVC also has a good ecosystem, although some of which is commercial. Other frameworks are quite specific to their task, but do have integrations with other projects.

Throughput/Scalability
Play was easily the winner here as it's features are so suited to scaling and throughput, supporting async applications with Akka actors. GWT and Vaadin also did well as their model is based around client side execution.

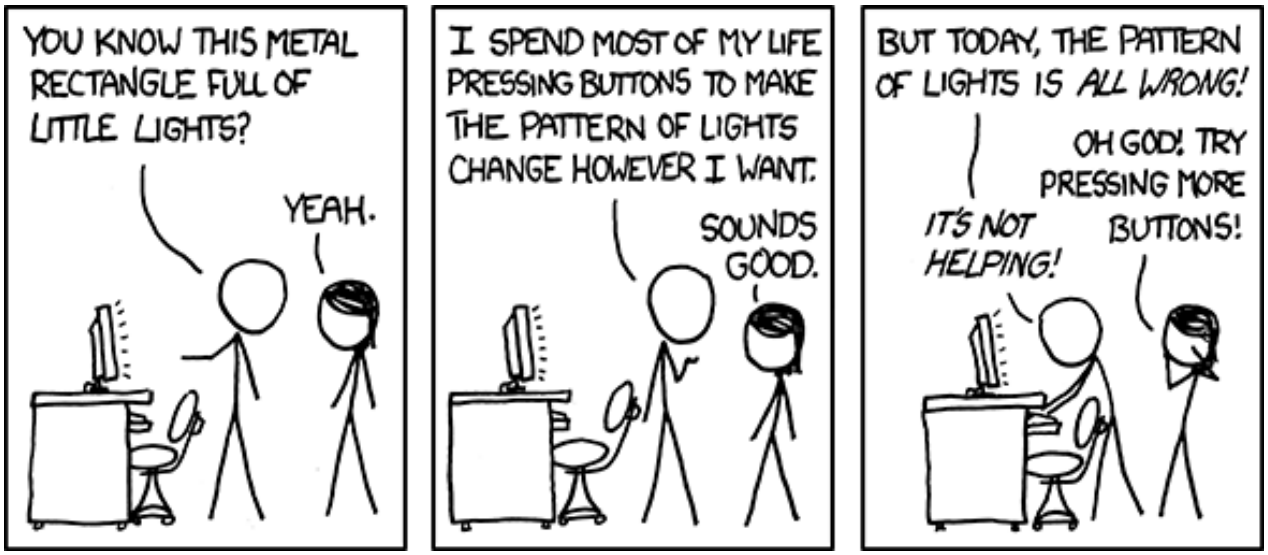
Code Maintenance/Updates
Grails and Wicket did particularly well with most config relying on default convention. Wicket also has a nice HTML/Java split which makes changes much easier.

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.

So what does this all mean?

As with our Great Java App Server debate, not all frameworks are created equal. In the same way that comparing WebLogic 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.

We're hoping to hear from product or team leads from each other frameworks, so we'll be able to hear some commentary from the horse's mouth. So stay tuned for that, and be sure to reach out to us in the comments section below, or ping @RebelLabs.



the comic: http://xkcd.com/722/

Contact Us

Twitter: @RebelLabs
Web: http://zeroturnaround.com/rebellabs
Email: labs@zeroturnaround.com

Estonia
Ülikooli 2, 5th floor
Tartu, Estonia, 51003
Phone: +372 740 4533

USA
545 Boylston St., 4th flr.
Boston, MA, USA, 02116
Phone: 1(857)277-1199

Czech Republic
Osadní 35 - Building B
Prague, Czech Republic 170 00
Phone: +372 740 4533

This report is brought to you by:
Simon Maple, Adam Koblentz, Sigmar Muuga, Jevgeni Martjushev, Cas Thomas, Sven Laanela, Oliver White, Ladislava Bohaova, Ryan St James