

Backbone

* Does not support observing functions
  + Any changes in properties need to be reset
  + Hard to test
* Imperative: tells machine how to do something, and what you want to happen
  + Define business logic
* Declarative: tell machine what you would like to happen, computer figure out how
  + User interfaces
* The case is that both of these are defined in the view. Thus no clear separation
  + Imperative programming, that is, to handle the event (so what happens when we get a new simulation, or when the simulation changes
  + Declarative programming, that is, listen to events and listen to change
* No data-binding
  + No sync between model and view
* Freedom to create and wire up components to your liking
* Have to write own boilerplate
* Need discipline for clean code
* Business logic are done through model and collections
* DOM construction is done via template engine. Manipulation done via data binding and imperative updates
* Stronger dependency for jQuery to do DOM manipulation
* Minimalistic and gives you freedom
  + That being said, you’ll have to write your own code
* As a common framework, many plugins have been created

Angular

* Distinction that declarative programming should be separate from imperative programming
* Business logic are plain simple JS object
* DOM construction is done in directive and does not separate construction from view logic
* View Logic
  + Declarative done by view
    - Like ng-repeat
    - These don’t need to be tested
  + Imperative done by controller
    - These should be tested
* Dependencies go in one direction, view to controller
  + Controller does not know of view. Thus can be reused
  + Backbone on the otherhand, uses Backbone.View and manipulates Dom nodes
* Provides built in model view syncing
  + Model is independent, does not know it’s being listened to
  + No need for dependency on observable properties
  + Functions are observables
  + Integrating 3rd party component requires added attention to ensure angular sees the changes those made on the model
  + Possible negative effects on performance, slow run time if too many controllers
* Does not require templating engine
  + Converts HTML to DSL
* Opinionated
  + Ex, no DOM manipulation in controllers because it affects testability
    - Use data-binding and directives for dom manipualation
* Few plugins as many required features are implemented already
* Require deeper knowledge of computer science
* Core feature is directives, which allows user to handle declarative programming. Shortfall is that writing directives are not easy

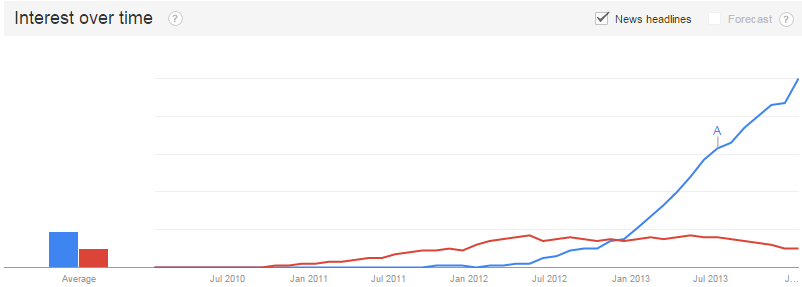
1.0 Introduction

The growing demand for dynamic web applications has driven an increase in new JavaScript web framework. The most popular type of framework follows an architect pattern called Model-View-Controller (MVC). This pattern addresses the issue of separation of concerns by splitting up the problem into three components: the model which contains the data, the view which contains the display and the controller which glues the model and view together. As this architecture helps solve various problems, many frameworks are beginning to implement the MVC pattern.

The increase in different frameworks has also led to a popular, yet never ending debate in finding the best framework. Although there is no concrete answer, comparing the different frameworks available will yield important findings. In this report, we will compare two popular frameworks, Backbone.js and AngularJS; the former of which is currently used by S&P Capital IQ. Firstly, we will provide a broad overview of the two different frameworks. We will then analyze the benefits of switching from Backbone.js to AngularJS. Finally we will provide solutions and methods to overcome the drawbacks that might occur from the migration.

2.0 Analysis

Both Backbone.js and AngularJS have been around for the same amount of time. However up until the beginning of 2013, Backbone.js was the more popular weapon of choice. As the year 2013 rolled in, Backbone.js received less activity while a huge interest for AngularJS occurred. Why did major shift in interest occur and should our company follow this trend?



As previously mentioned, many web application frameworks now a day follow the MVC pattern. Both Backbone.js and AngularJS implements if not closely, a variation of this pattern. In essence, each framework has their own idea on how to organize models, views and controllers. These different ideas are key criteria in creating a unique framework and are the reason behind the intense debates found online.

2.1.0 Backbone.js

To start off, Backbone.js is a small JavaScript framework that provides the basic structure for MVC. It relies on the Underscore library and often times, the jQuery library in order provide a complete platform for building web applications. Despite being the small framework that it is, Backbone.js offers a good set of conventions to help structure the code properly. This is done through the event-driven communication between Backbone’s model and views. One can attach an event listener to an attribute from the model and reflect the changes onto the view. Because of the lack in Backbone controllers (since Backbone views act as both views and controller), many programmers recognize Backbone.js as a framework that follows the Model-View-ViewModel (MVVM) pattern. Regardless of the title of the pattern, Backbone.js provides a respectable solution to organizing large and complex JavaScript code.

Another core feature of Backbone.js is that it is a small and minimalistic framework. This provides a lower learning curve for the user and at the same time, allow for clean and organized code. Furthermore, developers have the freedom to create and wire up their own components to their liking. If there are problems that cannot be solved with the Backbone framework alone, programmers can easily create their own plug-ins or include a third party library. That being the case, developers are often required to create the architecture and wiring themselves, which in the end, is time consuming and requires good attention to writing clean code.

2.1.1 AngularJS

On the other hand, AngularJS is a larger framework that was built to relief the problem of tightly coupled code between declarative programming with imperative programming. To understand the differences, imperative programming can be thought as “how to do something, and what you want to happen” while declarative programming can be thought as “what you would like to happen”. In Backbone.js, both of these programming styles are incorporated in the Backbone view. As a result there is a tight coupling between the DOM manipulation (manipulation of web elements) and the business logic. AngularJS solves this issue by enforcing declarative programming into angular directives and imperative programming into angular controllers. As a result, there exists a clear separation between the presentation and the data. Furthermore, AngularJS was created to encourage developers to follow a test-driven development process. In other words, developers should always think of different test cases first, before diving deep into writing code.

Unlike Backbone.js, AngularJS is extremely opinionated and requires developers to follow Angular’s way of development. Many programming practices derived from Backbone.js cannot be applied for development with AngularJS. For instance, creating user interface with JavaScript (often done by Backbone.js developers) is frowned upon when using AngularJS. As a result, a much steeper learning curve is required.

The syncing between models and views are also quite different when comparing Backbone.js with AngularJS. Instead of listening to changes from the model and the view, AngularJS has a built in data-binding system. AngularJS does this by extending the regular web flow by adding its own event loop, called the digest loop. It then identifies all bindings in the code and adds them into a watch list. The bindings are resolved (using a method called dirt checking) when the digest loop runs and any changes found are updated in the respective model or view. Since changes from the model can update the view, and vice versa, this method of data-binding is called two-way data-binding.

2.2 Advantages of switching frameworks

There are numerous advantages that can be gained from switching from a Backbone.js framework to an AngularJS framework. To begin with, AngularJS’ includes a large API that can be used to help simplify common tasks of a web application. For instance, loading a list of data and filtering them in real time can be done in one line (using Angular directives like ng-repeat and ng-filter). Being able to filter data in real time is also an impressive feature and is easily created by using Angular’s two-way data-binding system. In order to do the same thing, Backbone developers will need to attach event handlers (like on key up) to input boxes as well listeners to the model to listen to changes. Furthermore, each listen or change requires get and set functions respectively. This entire process is quite tedious since it is required each time a developer wants to sync up the model and the view.

Furthermore, the overall structure using AngularJS is reasonably clean and readable. This is because each component can easily be fit in the following types: Views (templates), Controllers, Factories, Services and Filters. Each of these can then be tested individually and re-used whenever necessary. On the other hand, Backbone.js only provides View, Model (Collections are similar to Models) and also depends on Underscore’s templating engine. Most of the logic are focused heavily on Backbone Views, and as a result, hinders the ability to re-use code.

Finally, the most important benefit that can be gained from AngularJS is that it enforces developer to write test-driven code. The ability to test individual modules (this process is called Unit-Testing) allows developers to gain confidence whenever editing existing code. This is especially useful in the case for large-scale project as many developers will be making changes to the project regularly. The ease in unit testing in AngularJS is created by a method called dependency injection. Each unit in the project is highly decoupled, separate from other modules and can be tested individually. In the event that modules depend on one another, testing can be done simply by injecting modules together.

2.3 Overcoming the difficulties

Although there are many advantages of AngularJS, switching frameworks can also yield some difficulties. As previously stated, AngularJS is a framework with a steeper learning curve. Switching from Backbone.js to AngularJS requires each developer to learn the new frameworks and adapt to the new programming styles. That being said, the growing community for AngularJS is an asset for developers in the event that they reach a difficult problem. However, once developers are familiar with this framework, an increase in productivity can be seen. Since much functionality are built into AngularJS, developers can save time writing tedious boilerplate code and spend more time creating features. The amount of time spent bug fixing will also be reduced as each modules are tested before making changes to the project.

As the programming practices between the two frameworks are quite different, migrating code will take a decent of time. Furthermore, things that are originally done in Backbone.js may not be the same as AngularJS. For instance, Backbone.js prefers rendering template and nest views to create the UI. AngularJS does not have a concrete way of doing this. Instead, this framework prefers to use one single view and is rendered automatically on Angular’s bootstrapping process. That being said, it is not impossible to nest views in AngularJS. The solution to this problem is to create angular directives. These directives act as templates and can provide added functionality. As a result, we can create one view and have nested directives which acts similar to Backbone’s nested views. Furthermore, this solution follows well with AngularJS’ coding practice to allow each unit (in this case, directives) to be easily tested. To see a concrete example, view Appendix 1. Another major difference between coding practices of the two frameworks is the ability to integrate third party library. Since Backbone.js is small and flexible framework, integrating third party library is very easy. For example, S&P Capital IQ relies on libraries like jQuery, slickgrid, Moment.js and many more. On the other hand, using third party library in AngularJS is much more difficult. These third party libraries will need to be converted into angular directives or services keeping testability in mind. Furthermore the amount of third-party library for AngularJS is much smaller than Backbone.js. However, this is not entirely a major concern as AngularJS is capable of replacing many libraries commonly used in Backbone.js. For instance, many functionally from the jQuery library are already a built in feature of AngularJS. The diagram on the side illustrates some of the common jQuery commands that are replaced by AngularJS. An example of incorporating jQuery into Angular can be found in Appendix 2

3.0 Conclusion

With so many frameworks available, web applications are evolving to become more interactive and more dynamic. Given each framework is distinct from one another, analyzing the benefits and downfall it comes with is always extremely important. Backbone.js is a lightweight framework giving developers the basic structure and foundation for clean and organize code. It furthermore allows developers the freedom to build upon the framework through a wide variety of third party libraries. As a result, small applications can be easily whipped up and maintained to your own liking. AnuglarJS, on the other hand, limits developer’s programming choices to follow the conventions provided by the framework. With the added ability of two-way data-binding, developers no long need to manually sync the model and view together. By forcing developers to split up large problems into decoupled and maintainable code, testing solutions has become easier than ever.

As the intense debates for the greatest framework continue, one can only question if an answer will ever exist. Instead of blindly following the trend, developers should pool in time to carefully research and analyze the frameworks that are best suited for situation. Each framework will have its own benefits and downfalls. The ability to adapt to these changes is the key in finding the most appropriate framework for the project.

http://jan.varwig.org/archive/angularjs-views-vs-directives

https://github.com/angular/angular.js/wiki/Understanding-Scopes#JSproto

http://stackoverflow.com/questions/14994391/thinking-in-angularjs-if-i-have-a-jquery-background

http://adamalbrecht.com/2013/12/12/creating-a-simple-modal-dialog-directive-in-angular-js/

http://jsfiddle.net/theparabolink/dnCHA/

Backbone vs angular

http://blog.nebithi.com/backbone-and-angular-demystifying-the-myths/

http://www.infoq.com/articles/backbone-vs-angular

https://www.airpair.com/js/javascript-framework-comparison

https://connect.liveperson.com/community/developers/blog/2014/02/04/my-own-angle-on-angularjs--pros-and-cons

http://www.quora.com/What-are-the-advantages-of-Back