new Views, you will likely find yourself inventing many of the pieces that make up an MV* framework.

At the outset, it isn't terribly difficult to write your own application framework that offers some opinionated way to avoid spaghetti code; however, to say that it is equally as trivial to write something as robust as Backbone would be a grossly incorrect assumption.

There's a lot more that goes into structuring an application than tying together a DOM manipulation library, templating, and routing. Mature MV* frameworks typically include not only the pieces you would find yourself writing, but also include solutions to problems you'll find yourself running into later on down the road. This is a time-saver that you shouldn't underestimate the value of.

So, where will you likely need an MV* framework and where won't you?

If you're writing an application where much of the heavy lifting for view rendering and data manipulation will be occurring in the browser, you may find a JavaScript MV* framework useful. Examples of applications that fall into this category are GMail, NewsBlur and the LinkedIn mobile app.

These types of applications typically download a single payload containing all the scripts, stylesheets, and markup users need for common tasks and then perform a lot of additional behavior in the background. For instance, it's trivial to switch between reading an email or document to writing one without sending a new page request to the server.

If, however, you're building an application that still relies on the server for most of the heavy-lifting of page/view rendering and you're just using a little JavaScript or jQuery to make things more interactive, an MV* framework may be overkill. There certainly are complex Web applications where the partial rendering of views can be coupled with a single-page application effectively, but for everything else, you may find yourself better sticking to a simpler setup.

Maturity in software (framework) development isn't simply about how long a framework has been around. It's about how solid the framework is and more importantly how well it's evolved to fill its role. Has it become more effective at solving common problems? Does it continue to improve as developers build larger and more complex applications with it?

## Why Consider Backbone.js?

Backbone provides a minimal set of data-structuring (Models, Collections) and user interface (Views, URLs) primitives that are helpful when building dynamic applications using JavaScript. It's not opinionated, meaning you have the freedom and flexibility to build the best experience for your web application how you see fit. You can either use

the prescribed architecture it offers out of the box or extend it to meet your requirements.

The library doesn't focus on widgets or replacing the way you structure objects—it just supplies you with utilities for manipulating and querying data in your application. It also doesn't prescribe a specific template engine—while you are free to use the Micro-templating offered by Underscore.js (one of it's dependencies), views can bind to HTML constructed using your templating solution of choice.

Looking at the large (*http://backbonejs.org/#examples*) number of applications built with Backbone, it's clear that it scales well. Backbone also works quite well with other libraries, meaning you can embed Backbone widgets in an application written with AngularJS, use it with TypeScript, or just use an individual class (like Models) as a data backer for simpler apps.

There are no performance drawbacks to using Backbone to structure your application. It avoids run loops, two-way binding, and constant polling of your data structures for updates and tries to keep things simple where possible. That said, should you wish to go against the grain, you can of course implement such things on top of it. Backbone won't stop you.

With a vibrant community of plugin and extension authors, there's a liklihood that if you're looking to achieve some behavior Backbone is lacking, a complementary project exists that works well with it. This is made simpler by Backbone offering literate documentation of it's source code, allowing anyone an opportunity to easily understand what is going on behind the scenes.

Having been refined over two and a half years of development, Backbone is a mature library that will continue to offer a minimalist solution for building better web applications. I regularly use it and hope that you find it as useful an addition to your toolbelt as I have.

## Setting Expectations

The goal of this book is to create an authoritative and centralized repository of information that can help those developing real-world apps with Backbone. If you come across a section or topic which you think could be improved or expanded on, please feel free to submit an issue (or better yet, a pull-request) on the book's GitHub site (*https://github.com/addyosmani/backbone-fundamentals*). It won't take long and you'll be helping other developers avoid the problems you ran into.

Topics will include MVC theory and how to build applications using Backbone's Models, Views, Collections, and Routers. I'll also be taking you through advanced topics like modular development with Backbone.js and AMD (via RequireJS), solutions to com-