AngularJS as an Introduction to Programming

Natalie Davenport

Advisor: Prof. Alyce Brady, Computer Science

A paper submitted in partial fulfillment of the requirements for the degree

of Bachelor of Arts at Kalamazoo College.

2015

Acknowledgments

There are many who contributed to this project. First, I would like to thank Professor Alyce Brady, my SIP advisor, who not only helped me decide on my topic, but also offered encouragements and general life advice.

Next, I would like to thank Professor Pamela Cutter, who led our department's SIP seminar and kept our group on track during the writing and revision process.
I also thank my peers who read my SIP and offered helpful feedback. Special thanks goes to Marc Kuniansky who read my SIP in its near-entirety, offering extensive advice which greatly improved the quality of this paper.

Finally, I would like to thank the Monroe-Brown Foundation for offering me this excellent internship, as well as the great people at BASIC who served as my mentors over the course of the summer.

Table of Contents

Introduction

Over the course of this summer, I worked as a programming intern at BASIC, a human resources solutions company. One of my projects required that I learn AngularJS, a Javascript framework written to assist in the creation of web applications. I was struck by Angular's core concepts, as well as the resources available to learn it. After this experience, I decided to evaluate Angular with the intention of discussing its use as a tool to introduce others to programming and web development. In recent years, Javascript has undergone a transformation, resulting in a strong culture of open source Javascript tools written for developers. In this context, I also discuss the importance of learning Javascript as a programming language, not only for use in webpages. In order to do this, I apply the principles of Angular to the activities of CS105: Introduction to Computer Science, which is Kalamazoo College's gateway course to the Computer Science department, to demonstrate how it could be incorporated into its curriculum.

(At the time of this writing, Angular 2 is currently in beta, bringing a complete rewrite of Angular 1. According to members of the Angular team, Angular 2 bears little resemble to its predecessor. In the face of skepticism of this next version, the team has also made it clear that support for Angular 1 will continue. There also exists a means to migrate Angular 1 code to Angular 2 using ng-Upgrade. Over this summer, I worked with Angular 1.4, so it is used as the basis of this paper's discussion.)

Part 1: The Internship

I. About the Internship

Over the summer, I was employed as an intern at BASIC (not an acronym), a third-party administrator specializing in Flexible Spending Accounts (FSA). BASIC administers human-resources services, including COBRA, payroll, and other forms of health-savings accounts. While I did not set out with a particularly strong idea of what sort of company I wanted to intern at, I did not anticipate a company like BASIC until the position appeared. Going into it, I had little idea about what sort of software I would be working on, knowing even less about the behind-the-scenes operations of such companies.

The position was described as an internship in the IT department, stating only that an array of projects requiring programming and software design skills were available. My primary criteria for a summer internship was for a position that would provide me with hands-on experience and a chance to apply the skills I already possessed. At that point, I was not confident in my programming abilities, and was uncertain as to whether I would be able to apply the theory I had learned from my classes at Kalamazoo College.

I began working at BASIC part-time in May of 2015, balancing 12-hour weeks with my academic schedule. I was to work with a group of four people who made up the IT and programing department. Three members of this team handled infrastructure, managing the databases, networking, and office technology. The other member of the department held the title of “Senior Programmer” and was responsible for writing and

maintaining most of the company's active code. Over the years, the department had a rotating pool of interns and contractors, but never more than two dedicated programmers.

On my first day at BASIC, my supervisor walked me through the company's services and everything my position would encompass over the summer. This included BASIC's active products, background on the company, and the department's open projects. This tour also included a survey of BASIC's servers and data storage. In the past, BASIC had subcontracted the storage of its data to an outside group. Once BASIC reclaimed control over these services, the data was never reintegrated in a cohesive manner, adding another layer of complexity to the way that BASIC's data was presented. In this huge wave of information, I was introduced to many of the projects on which I would later spend my summer.

My supervisor served as my mentor, assigning me to the tasks and projects he thought would fit my skills. I was left with busy work at times, due to his hectic schedule and his role as the only official programmer at BASIC. However, the majority of my time was spent working on projects which allowed me to research many new topics. One advantage to the small programming department was that these projects related to services that BASIC offered and then were destined to be deployed. In order to prepare for these assignments, my supervisor sent me a list of skills and technologies to learn that would be necessary for that particular project and gave me a few days to familiarize myself with the framework or language. This meant sorting my way through a lengthy stream of articles and tutorials on the subject, usually with little feedback on the quality of my code or the many errors I encountered. While research was slow at times, halted by

bugs in my code, it also meant that my ability to pinpoint the weak spots of my code improved.

II. Projects and Responsibilities

For my major projects of the summer, my supervisor and I used Team Foundation Server (TFS) as our version control software. The two largest projects, both in Visual Studio, integrated quite easily with the built-in team extension. Before the summer, my only experience with version control had been with a small student group, using Git and GitHub, so I welcomed the additional exposure.

My initial workload was small; the first tasks were simple reports for both internal and external use. As the summer progressed, I worked on larger projects. These included revising an internal Java application and two large web applications, which introduced me to both C# and AngularJS.

A. Reports

My first task was to create reports, online presentations of data queried from BASIC's database, that would be used both inside and outside BASIC. At the core of a report was the query to the database, written using Transactional SQL (T-SQL), Microsoft's brand of Structured Query Language (SQL), designed for use with SQL Server. I taught myself the relevant skills by reading a T-SQL manual, tutorials online, and old reports. To write the query and build the report, I relied heavily on SQL Server Management Studio (SSMS), which included tools to interact with the server and visually build the reports. I worked on a total of five reports of varying complexity, each of these taking one to three days. BASIC relies on SQL Server 2000, which limits some of the

technology that the company currently uses. There are more modern alternatives for several of the programs in SSMS, but BASIC was working on migrating to new server software. Overall, this was a good way of not only familiarizing myself with SSMS and accessing the servers, but also with BASIC's data and services. After the third or fourth report, the tasks took on the feeling of busy work. Fortunately, my supervisor quickly gave me new assignments.

B. Contact Sync Program

My second project was the Contact Sync program, which updated contacts and personal information from an outside service to BASIC's contact database. Unlike many other active projects at BASIC, this program was written in Java. It was delegated to me because I had the most experience in the language. Another intern who had written the application several years before had handled this goal by connecting to the external service's API and then using a Java-SQL library to handle interactions with BASIC's server. When my supervisor gave me this project, he mentioned that the ultimate goal was to create a program that could be run daily on the server, which would send alerts to IT when it encountered issues. My supervisor also mentioned that the intern who had written the program had been unable to get it running remotely, and thus had to run it manually.

My initial goal was to build the program to a JAR and then set it up to run automatically. However, after viewing the program's code, it became an exercise in software design. Most of my time spent on this project went into reorganizing the code into appropriate classes and methods, as well as implementing an improved logging

system. While I already knew Java, I also researched proper software design patterns that would be effective in this project. For logging, I used a popular logging library, Log4J, something I ended up using later in the summer. I included lines invoking the logger throughout the application, especially in areas of the code with the highest potential for failure. After I spent several weeks on this project, the program was able to be automatically run from the server.

C. Web IVR

While I did learn valuable information from my two previous tasks, my largest project was Web IVR, BASIC's online interactive voice response program. My supervisor had already put significant time and effort into the project and planned to deploy it after the end of the summer. The Web IVR portal provides a way for employees to enter their absences and missed times online instead of over the phone. The project had two sides: the employee side and the management side. The employee side presented a tree of questions that would determine the nature of the absence based on the employee's information, then registering that info with the appropriate parties. The management side gave managers a way to edit the question tree and access the data their employees had submitted. This also required a ticket system to keep track of bugs found in the service, as well as a way to sort through and display these tickets on the development end.

Web IVR required many concepts that I had barely heard of before working at BASIC. The entire project was housed in Visual Studio, a powerful design environment with which I had no previous experience. Because of the interconnected nature of Microsoft's products, the project was cohesively built with tools offered in Visual Studio.

My biggest challenge was navigating its structure and discerning the relationships of the components. Like the other projects I handled this summer, I dove into Web IVR with only a basic knowledge of its features and languages and then continued with my own research.

For the backend of the project, we used ASP.NET Web API, which is a RESTful framework that handles web transactions for applications. REST, or representational state transfer, refers to a style of application that hinges on the statelessness of its HTTP calls. While Web API streamlines certain aspects of RESTful services, one of its strongest points is that it doesn't completely abstract the HTTP responses from the developer, making interactions with the developed API more customizable. As part of Microsoft's large library of code, Web API is written in C#. Behind Web API was Entity Framework (EF), another Microsoft code base. Entity Framework, also written in C#, supplies a means for .NET applications to interact with relational databases. This combination of Web API and EF facilitated my use of BASIC's data, also acting as a convenient introduction to Visual Studio and the .NET platform.

We used AngularJS, a popular Javascript framework, for the frontend of the project. When combined with Bootstrap, another framework that streamlines page styling, constructing dynamic web pages was made far easier. Angular structured the app on the frontend, as well as the interactions with the backend of the project. In addition to Angular and Bootstrap, the project also required Node Package Manager (NPM), Grunt, Bower, Karma, Jasmine, and D3. Unfortunately, I didn't have much of a chance to work with D3, the visualization library, but did spend time with the others.

At first, I expected my rusty knowledge of Javascript to present the biggest challenge. I actually discovered my roadblock was understanding the structure of an Angular app. Angular's strength isn't that it adds additional functions to straight Javascript, as with many other libraries, but that it provides an overall architecture for an app.

These came together to form a single-page app, which is a web application that builds the content dynamically as the user interacts with the page and the API. For our team, the biggest advantages of this line-up was the faster loading time.

Like the previous projects, I dove right into work on Web IVR with no pre-existing knowledge of Web API, EF, or Angular. The biggest challenge was figuring out how to navigate the project structure, determining what was in control of the frontend vs the backend in particular. By far, it took the longest time to familiarize myself with Web API and EF. As these frameworks are in wide use, it was easy to locate tutorials and resources. I began with minor graphics fixes to pre-existing pages, then moved on to creating a system to manage user-submitted tickets. I spent most of my time on this project implementing this feature. Additional effort also went into refactoring existing parts of the app.

D. Forms Project

My second largest project was the forms project. In many ways, this project was similar to Web IVR in that it used Web API, EF, and Angular to build a web application. However, this project was primarily my responsibility, so as a result, I handled the bulk of its set-up. The web application was an online version of an external form that BASIC

often sent out to individuals. In order to digitize the process, there needed to be an online page to enter the data, as well as the appropriate tables in the database to store it. Creating the database and the tables was also my responsibility. Due to the scale of the project, the structure of the code for both Web API and Angular files had to be altered to accommodate future forms.

I developed this project faster than Web IVR because I had a deeper understanding of everything used. This project also required several features that were completely new to me. First, it needed the capability to convert the data into a PDF, which required working with an unfamiliar but extremely interesting library written in C#. Another was email capability, achieved with standard C# libraries.

I also had to add behavior-driven tests for the application's Angular code. For this, I used a combination of Jasmine, an elegant testing library, and Karma, a test runner, often used specifically for Angular web apps. Using these together, I was able to implement proper behavior-driven tests.

III. Challenges

Hands-off mentoring has its drawbacks. While tracking down the cause of a troublesome bug is a rewarding experience, it has significant impacts on efficiency. Many of the errors I encountered could have been fixed by consulting my mentor, but I had to instead resolve them through hours of frustrating searching. I documented my process and research in daily notes. With these, I could quickly refer back to the topics and interesting facts I encountered each day. Nearly all of the errors I encountered are documented, which often saved me from making the same mistake twice.

As the only new addition to the team with fewer than 10 years experience on the job, I often found myself scrambling even to understand the projects I was assigned. Looking back, my experience would have been significantly easier with more frequent, easily acquired feedback. Nearing the end of the summer, BASIC hired on another programmer to assist my supervisor with his projects. During this addition, my supervisor and the new programmer moved their workstations to the area in which I had been working. With this change, the pace of my research immediately sped up, unfortunate only in that it occurred so late into the summer.

I also experienced difficulty in a social sense. In terms of placement, my workstation was isolated not only from the rest of the IT department, but also from the rest of the office. Starting on the first day, I was stationed by myself in a cubicle in the corner of BASIC's main floor. This meant that I had little chance to meet others in the office, as there was limited foot traffic through the corner.

Another challenge I encountered during my time at BASIC was the extreme gender inequality in the IT department. The lack of gender diversity in the field of Computer Science is already a well-known issue.[1] The effects of this phenomenon appear in subtle but nonetheless uncomfortable ways. Not only was I the only woman in the IT department, but I was also the first female intern that had come to BASIC for that internship, an unfortunate fact that did not surprise me. While I experienced no outright sexist episodes, this discrepancy manifested in other ways.

---

[1] Robertson, Judy. "Girls Can't Program in Their Heads: Gender and Games in the Computing Classroom." *Communications of the ACM*. June 29, 2012. Accessed December 4, 2015.

It added an undeniable edge to my isolation. In some ways, I wonder if I would have experienced the same hands-off mentoring style if I were male. If it had been easier for me to connect with the other members of the IT department, it would have been easier to ask for advice. During my research, there were many times in which I encountered problems I had no idea how to solve, putting me at a loss and blocking my progress for days at a time. For nearly all of these, I struggled on my own, rather than asking for help. In a few of these cases, when I eventually did ask for assistance from my supervisor, the problem usually turned out to have been a minor error, and was explained in a matter of minutes. However, because of my fear of being seen as inexperienced, I didn't feel comfortable approaching them unless absolutely necessary. While this is also a product of limited real-world experience, it is also inextricably linked to my status as a minority within the field. At times, as the only woman, I felt as though my abilities as a programmer were a reflection on my gender, rather than my individual skill.

Oftentimes, the barriers that I and other women face do not take the form of traditional sexism. In this case, my work is inextricably bound with my gender, as the workplace will never allow me to separate my ability from my identity. This is not the direct fault of anyone at BASIC, but it is a symptom of a larger problem. This issue is documented across nearly all male-dominated fields. In an article discussing the nature of the "confidence gap", authors Katty Kay and Claire Shipman partially attribute its effect on women in the industry to the field's inequality: "There is a particular crisis for women—a vast confidence gap that separates the sexes. Compared with men, women don't consider themselves as ready for promotions, they predict they'll do worse on tests,

and they generally underestimate their abilities."[2] This lack of strong feedback definitely impacted my enthusiasm and quality of my work, out of fear of setting back the progress of our team.

I know that I am not the only woman to have experienced situations like these. While it may not have been the fault of the great people at BASIC, it is an indication of a harmful, multifaceted imbalance that faces Computer Science as a whole. For anyone who does not fit comfortably into the standard profile of 'programmer', it is critical to have mentors within the field with similar identities, both for personal support and as proof that success is possible in such homogeneity.

IV. Summary

BASIC gave me the opportunity to apply my Computer Science education from Kalamazoo to a professional programming environment. I can divide what I gained this summer into two categories: team experience and technical ability.

The former includes my experience with version control and working with and revising other programmers' code. In order to do this, I had to be able to understand unfamiliar code and recognize the areas where it could be improved. This also meant that I had to grow accustomed to the regular use of version control, which is a critical experience to have in any development environment.

This was experience that I had to gain outside of Kalamazoo College to develop good programming habits. While the teaching and learning of theory and practice is

[2] Kay, Katty, and Claire Shipman. "The Confidence Gap." *The Atlantic*. April 14, 2014. Accessed November 11, 2015.

valuable, the application of this knowledge is absolutely required. Unfortunately, Kalamazoo College lacks the in-depth and special topics courses that would help to remedy this problem. However, it does provide its students with the skills to research and quickly acquire new languages and concepts.

Out of everything I learned at BASIC, Angular was the most valuable. Using Angular, I was able to quickly learn web development principles, all while adding a powerful item to my programming arsenal. Not only was it pivotal in my two largest applications, but also has since proven useful in my independant projects. It stands out through its potential as an introductory path into web development. Angular goes beyond a collection of functions and data to add to an existing project, and instead provides an all-encompassing structure around which the app is based. It gives developers a way to implement popular design principles while providing the means to do so in an organized fashion. Results on the page are easy to acquire after set-up is finished, making it a strong contender for an introductory website building tool.

In the rest of this paper, I will discuss Angular's strengths and weaknesses, as well as its value, both in a broader sense and as part of Kalamazoo College's Computer Science curriculum.

Part 2: AngularJS

I. The Evolution of Javascript

Finding a reliable way to work with the barebones of a webpage has long been a challenge for developers. The HTML DOM, or Document Object Model, which is the visual markup of a page as described by HTML, presents unique problems for the representation of dynamic content. Working with HTML can be tedious and impractical. In order to create the animated and interactive content that characterizes so many of today's best websites, developers must extend their reach beyond HTML and plain Javascript. Jquery has filled this position since the early 2000s, but this has its limits as a library. JQuery's primary services are DOM traversal and manipulation, event handling and animation.

Especially within the last decade, the number of open source Javascript tools available for developers has skyrocketed. Part of the attraction of these tools has been that they offer the uniformity of Javascript through the entire project. Including additional libraries of code has become trivially easy, and with the right package manager, requires only an extra line or two within the application. Many web developers no longer code with just 'vanilla' Javascript — code written without the use of additional frameworks. In fact, the prevalence of frameworks has created a popular gag: the existence of VanillaJS, the most lightweight JS library in existence.[3]

As suggested in an article written by Scott Hanselman, well-known developer for Microsoft, Javascript might now be the assembly code of the web. It has even been used outside of browser environments, supporting the idea of "full-stack Javascript", where

[3] Wastl, Eric. "Vanilla JS." Vanilla JS. Accessed November 12, 2015. http://vanilla-js.com/.

Javascript is used for all components of a web application, including any server side code and the software for the database itself.[4]

In this way, Javascript could be seen as reaching its true potential through the construction of larger tools and frameworks. In fact, as of November 9th, in a ranking of the most popular Javascript projects hosted on GitHub, most of the top repositories are frameworks intended to be used by developers. It is in this context that Angular has achieved its following over the last several years

II. About Angular

Angular, originally entitled "GetAngular", was the side-project of two Google engineers, Misko Hevery and Adam Abrons. First used in 2009 to rewrite Google Feedback, Angular provided a way to translate the application's 17,000+ lines of code into an Angular app only 1,500 lines long in just 3 weeks. Despite some initial hesitation from other Google engineers, Angular was adopted by other Googlers. From that point, its user base began to expand, and in 2013, the framework's popularity exploded.[5]

It is the most-searched Javascript framework of its kind, in use by indie developers to large enterprises.[6] In addition to a huge base of active users, there are meetups and training sessions for Angular developers, as well as Ng-Conf, a conference focusing exclusively on Angular. Angular's ability to address the needs of web

[4] Hanselman, Scott. "JavaScript Is Web Assembly Language and That's OK." Scott Hanselman. May 24, 2013. Accessed November 12, 2015.

[5] Hevery, Miško, and Brad Green. "Keynote - NG-Conf 2014." *YouTube*. January 16, 2014. Accessed December 4, 2015. https://youtu.be/r1A1VR0ibIQ.

[6] Borlinghaus, Tobi, and Daniel Clasen. "Comparison of 4 Popular JavaScript MV* Frameworks (part 2)." *Developer Economics.* March 5, 2015. Accessed November 11, 2015.

developers was its cause for success. Not only is it customizable, but also fast and modular.

With the Angular framework, nearly all Javascript code is encompassed within the Angular app structure. As opposed to libraries, which is a collection of additional tools that can be plugged into a project to expand an existing code base, a framework provides additional code, but also affects how the code within a project is presented.

While Angular is far from the only framework of its kind, it stands out in its handling of two-way databinding. This refers to the binding between the user interface (UI) on the page and the data model. Elements incorporated directly into the HTML of a page are bound to the data, so any changes detected on the page are immediately updated.

This has huge implications for developers, and allows them to work with the UI in ways that would otherwise be difficult. The prevalence of single-page apps is only possible through frameworks like Angular that allow this sort of UI manipulation. There are many advantages for building apps in this fashion. Loading time for new components of a page are dramatically reduced, as a new HTML file no longer has to load with each interaction or change of page. Navigation is also streamlined and handled through the Angular core of the app. The use of Angular also enables commonly used tools, like form validation and paging operations. Interaction with backend services is also incorporated in Angular through $http and $resource, services that handle calls to web services and databases.

III. Notable Users

Although Angular is a tool used by many indie web developers, they do not make up the entirety of its user base. In fact, many prominent companies have opted to use Angular as the framework of choice, some since its earliest days. Obviously, in recognition of its origins, Google is one of these users; others include Itunes Connect, CNN, MSNBC, Virgin America, and Amazon.[7]

Using Angular benefits larger companies in multiple ways. As a project backed by Google, Angular comes with a status of credibility and guaranteed support. With smaller frameworks, there exists the worry that their development teams will move on to other projects. In the case of Angular, there is a dedicated team within Google focused on maintaining the code. In addition, as an open source project, Angular also is distributed under MIT's open license, which means it's not subject to the same restrictions as its proprietary alternatives.

Single-page apps are now the standard for well-designed websites. They come with the advantage of handling large numbers of pages and significant traffic. Also, Angular modules are highly testable. Angular itself has testing frameworks designed specifically for it, which itself is a huge attraction for many users. In addition, because of its flexibility, Angular can be applied to a number of problems, making it applicable to many different web services belonging to one company.

[7] Polepeddi, Lalith. "Made with Angular." *Made with Angular*. Accessed December 4, 2015. https://www.madewithangular.com/#/.

With the endorsements of these large companies, Angular is guaranteed its place as one of the go-to frameworks for web applications, indicating that Angular isn't going anywhere.

IV. Core Concepts of Angular

Angular uses the Model-View-ViewModel (MVVM) architecture to organize an application, which is a variety of Model-View-Controller (MVC). The model is the form taken by the user's data representing the desired information. Model classes are most often used to correlate to representations in a database. The view refers to the side of the application available to the user, and controls how the user interacts with or enters data. Neither the model nor the view are the actual code necessary to manipulate the data. Instead, this is left to the controller, which encompasses the functionality of the app, incorporating any business logic. However, in MVVM, the ViewModel accounts for UI binding. Changes in the model can result in changes to how the data is presented, so the ViewModel handles this as well as the logic of the app. These changes are registered with $digest, which monitors the given fields and notifies the controller of changes by running a loop checking for updated values.

V. Parts of the Code

As a framework, Angular comes with different types of components that are used throughout the app. Out of its many parts, the most important are as explained below.

A. Controllers

Controllers are the central part of an Angular app, supplying a page's behavior. While not contributing to the structure of the data received from the user, the controller

handles the logic behind the binding of the data to the model. At minimum, there is one controller per Angular app, sometimes defined implicitly. Controllers are most often assigned from within an ng-Route module, rather than with the ng-controller directive. Through controllers, the functionality of other Angular components is combined into the page.

B. Services

Unfortunately, naming conventions are not one of Angular's strong areas, so there exist multiple types featuring the same name. In the most common sense, however, 'service' designates a type of Angular class that provides logic to manipulate the data model.

Services are commonly designed as singletons, meaning that they are created once and are injected as needed into other parts of the app. There are three important subclasses of service: factories, providers, and services. Factories return copies or instances of a given object, often containing some form of a 'createObject' method. Services generate instances of objects by using the 'new' keyword, like a direct constructor. Providers are similar to services, but are unique in the fact that they can interact with config modules, a special type of Angular module. Providers also return their data through the use of the '$get' function, which itself is a service.

These distinctions are important because of the frequency of services in Angular code, especially within larger applications.

C. Models

While services contain the business logic of a feature, models hold the form of the user's data. Model classes should not include any of the logic controlled by services, and instead only represent the barebones data structure for an object. Most often, the form of the model corresponds to its structure within a database.

D. Directives

While models and controllers appear in other languages, directives are what distinguishes Angular from other frameworks. They bind fragments of code and HTML to the page through the use of scope. They are very powerful tools, as they are under control of the page's controller and are reusable across the entire application. They can easily be added to pages by defining custom HTML tags, and can serve many roles — custom input elements, video controllers, data display, and much more.

E. Modules

Modules are used to organize and contain code within a project. A module is injected into another module for use by simply listing its name within the client module's declaration as a dependency. This is also the same fashion in which outside modules are linked to a project. One of the most important external modules is ng-Route, which provides a routing service to an app, allowing for seamless transitions without navigation between entire HTML documents. In addition, the use of modules promotes proper project structure and control over the application-wide scope and reduces the potential for variable pollution.

VI. Simple Angular Apps

Including Angular in a project can be done with two lines. First, a script tag linking to Angular's source code must be included somewhere on the main HTML page of the project, either to the online version of the code, or to its location in the project's library. Second, the ng-app directive must be added to an HTML tag within the document, usually in the body tag. Any Angular code within that element will then be rendered on the page.

With the right tools it's possible to construct the skeleton of an Angular app in minutes, even with complex chains of dependencies. This process is easiest when using the terminal or command line. First, Node Package Manager(npm) must be installed to use Bower, one of the simplest frontend package managers best suited to Angular. Bower is perfect for managing the dependencies for Angular and Angular-related libraries, such as ng-Route, ng-Resource, and Bootstrap. Using Bower, it's possible to automate the most frustrating part of the project in order to move on to actual web development. At BASIC, we used this process to coordinate all of the dependencies and source files within our projects.

The simplest way to display data on the page is with Angular Expressions, which appear as a set of double curly-brackets, the contents of which will be evaluated by Angular. This is an easy way to tell if Angular is correctly bound to the project. Below is an example of a basic Angular app. With this page, an indirect controller and module are tied directly to the DOM with ng-app and ng-model.

```
<!DOCTYPE html>
<head>
	<title>Super Simple Angular App</title>
	<script src="bower_components/angular/angular.min.js"></script>
</head>
<body ng-app>
	<div>
		<label>Name:</label>
		<input type="text" ng-model="yourName" placeholder="Your
	name">
		<h1>Hello {{yourName}}!</h1>
	</div>
</body>
</html>
```

Following convention, Angular apps should display at least basic code organization. Below is an example of a simple controller, bound to the page with the ng-controller directive. In this example, the sum of the parsed values of two input fields is updated in real time as the user types.

Javascript:

```
(function(){
	angular.module('app', [])
	.controller('mainController', function($scope){
		$scope.addNumbers = function(num1, num2){
			$scope.result = parseInt(num1) + parseInt(num2);
		}
	});
})();
```

HTML:

```
<!DOCTYPE html>
<head>
	<title>Sample Angular Page</title>
	<script type="text/javascript" src="Your Angular src file">
		</script>
	<script type="text/javascript" src="YourJSCode.js" ></script>
</head>
<body ng-app="app" ng-controller="mainController">
	<div>
		<input type="text" ng-model="number1">
		<input type="text" ng-model="number2">
		<button ng-click="addNumbers(number1, number2)">Add</button>
		<p>The result of {{number1}} + {{number2}} is {{result}}</p>
	</div>
</body>
</html>
```

The Angular webpage features templates for simple applications. While there exists an official Google style guide, it lacks detail. In its place are decisive style guides written by the forerunners of the Angular community. John Papa and Todd Motto, two well-known web developers, have published their own popular style guides, both of which are endorsed by the Angular team as a verbose and practical alternative.[8] [9]

Most style guides suggest that a project's javascript be grouped by component type. For example, controllers, models, and services often constitute their own subsections. This form applies to the majority of small or medium apps. In the case of our Web IVR project, there were too many files to follow this structure. Instead, the project was designed around an alternative style grouping files by page and then by type.

VII. Challenges of Angular

Like any other framework, Angular comes with its own challenges. One of its well-known flaws is the quality of its documentation. The docs inadequately describe Angular's features, and as many members of the community have frequently discussed, are incomplete and even inaccurate in places. In addition, the descriptions are riddled with obtuse language, which simply isn't accessible to those encountering Angular for the first time. This is especially strange, given the number of self-taught developers using Angular as their framework of choice.

As a flexible framework, there are many ways to accomplish a single task, which doesn't help those just starting out. There exists a steep learning for Angular's more

[8] Motto, Todd. "AngularJS Style Guide." GitHub. July 27, 2014. Accessed November 11, 2015. https://github.com/toddmotto/angularjs-styleguide.

[9] Papa, John. "Angular Style Guide." *GitHub*. July 28, 2014. Accessed November 11, 2015. https://github.com/johnpapa/angular-styleguide.

complex services, like $resource or the promise library. Managing dependencies annoy even Angular veterans at times, especially in complex projects.

Error messages produced by Angular code are cryptic, rarely indicating the problem. Unfortunately, dependency-related errors are not accompanied by helpful messages, as I discovered while working with modules.

VIII. The Angular Community

Despite Angular's weaknesses, the community fosters a strong culture of learning. It's extremely easy to find tutorials on particular topics, many of which are written in step-by-step style for those with less technical background. Videos, podcasts, and even in-person gatherings supplement the expected array of blog posts and articles, increasing the range of accessibility. Advanced and intermediate topics are interspersed with the expected number of Angular 101 entries. In many cases, the authors assume little about the ability of their audience, which is part of the reason Angular attracts first-time web developers.

The Angular community also spends time adding to the code and support documentation. Widespread popularity has also lead to the creation of Ng-Conf, a conference dedicated to developing in Angular, which first took place in 2014 in Salt Lake City. Contributors also play a significant role in the future steps of the framework, with user feedback heavily influencing the design of Angular 2 and the continued support for Angular 1.

IX. Angular as a Way into Web Development

From my experience over the summer, learning Angular is not especially difficult. There is an undeniable learning curve that begins at some point shortly after the introduction of services and modules, but is not insurmountable. As with the majority of programming languages and design patterns, sufficient instruction and resources overcome conceptual difficulties. Fortunately, the body of articles, guides, and tutorials produced by the Angular community allows anyone, no matter their background in programming or web design, to construct and manipulate web page content. However, Angular would have presented a greater challenge if I didn't have a strong background in object oriented design and rudimentary web development knowledge. Clearly there have been many self-taught developers who have successfully navigated this issue.

In this context, how effective would Angular be as an introduction to programming for those with little to no previous experience? In a setting where Javascript and HTML alone are obsolete, could Angular be used to bring curriculums up to modern standards? The next section will discuss the potential value of Angular and other Javascript frameworks as an introduction to programming.

Part 3: Angular as an Introduction to Programming

I. Learning Angular

I first began to consider Angular as a potential tool for teaching when I started working on the frontend of Web IVR. During my research, I was struck by how easy it was to learn Angular simply by following tutorials written by the community. Given that I had several years of programming experience, it wasn't surprising that certain concepts — objects, functions, reusability — helped in this process. However, I had never worked with Javascript beyond a purely introductory level. My only experience had come from the college's most basic computer science course, CS105: Introduction to Computer Science. Despite this, it took only several weeks before I was comfortable developing complex behaviors with Angular. I found the framework powerful and intuitive, and many things that might have delayed my work in plain Javascript — working with the DOM — were avoided by UI binding and strict app structure.

This code was easily integrated into the other parts of the project. With $resource, Angular's service for handling database connections and API calls, integrating the Web API backend was simple. After this, I applied this knowledge to my other projects at BASIC, and then even my own side projects.

To my surprise, most of the tutorials I used were written for an audience of complete beginners. Even for complex topics, the authors went to great lengths to make their guides beginner-friendly, providing resources and links for referenced concepts.

Since then, my experience with Angular has helped me to pick up other frameworks, serving as a great introduction to web development on the whole. These

experiences prompted me to examine Angular as an entrance point to programming, and in turn, how it would integrate with the Computer Science curriculum.

II. Implications for CS105

Currently, web development is found only in K's most basic introductory computer science course, CS105. In past sessions, students have spent the first two thirds of the class doing small-scale Javascript and HTML assignments, never branching beyond the most basic functionality of either tool. It is in this place that more developed material could be introduced.

Javascript is used to teach core ideas of programming, in particular, variables, functions, data types, and simple objects. Beyond this, web development does not make another appearance in the curriculum, which is surprising, given the size of the industry. There are two possible options for updating the curriculum to include web development: first, bring the content of the class up to date by using modern frameworks. These core programming concepts could still be conveyed, but also with the instruction of a practical and useful framework. Second, more special topics courses focusing on web development and modern tools could be introduced.

III. CS105 Assignments Written in Angular

To gauge what this would look like, I assessed the programming assignments of CS105, considering the lessons each assignment is designed to teach. I then implemented these using Angular. For the visual component, I used BootStrap, which is industry standard. I selected these activities for the concepts they teach, in order to determine how well Angular preserved these goals.

A. The Activities

One of the earliest mini-labs is the Lucky Button assignment, in which the student constructs a page with a button calling a method to generate a random number and 3-letter word, which is then displayed on the page. This assignment is intended to demonstrate buttons, functions, string manipulation, and the Javascript Math library. The goals of this assignment are preserved in the Angular implementation. Although by means of a slightly different directive, the functions are linked to the buttons on the page, and the content is still acquired through the use of the Math library and string manipulation.

Following this is the Virtual Pet project and lab. In this assignment, the student is asked to program a set of states that change based on a timer function and random possibility. Working on the same principles as Lucky Button, it requires knowledge of HTML elements, functions, and the Math library. However, this assignment is more complex, requiring multiple functions and if-statements. The sample Angular implementation demonstrates how the framework could be used to focus the student on the Javascript functions, rather than on the HTML.

The Address List lab and programming project explores arrays, for-loops, simple objects, and sorting. If fully implemented in Angular, almost all of these concepts would be preserved. Ng-repeat, a directive used in an HTML element which generates copies of that element based on a list of Javascript items, eliminates the need for a for-loop. However, this feature could easily be implemented as originally intended using a loop. Similar to the Virtual Pet rewrite, Angular removes the need to work directly with the

HTML elements, letting the student focus on important concepts, such as sorting and selecting from an array.

The final activity I rewrote was the Message Board lab, which was chosen to demonstrate dynamic styling and Angular's elegant handling of form elements. This was incredibly easy to implement, as there is virtually no code required. A preview of the message is generated and the necessary values are pulled directly from the input elements of the page through UI binding, a difficult task without Angular.

B. The Rewrites

For the most part, the programming concepts of these lessons were preserved in the Angular rewrite. The resulting changes are as follows: the Javascript, not the HTML, holds the focus and the resulting code is both cleaner and simpler without DOM manipulation. A link to the code is available in the Appendix of this paper.

IV. Advantages and Disadvantages

There are some drawbacks to using Angular as an introductory tool. First, project set-up is a notoriously big step for Javascript novices. Constructing a working system of modules can be tricky with no previous experience. However, with step-by-step assistance and a few examples, this is an easy issue to avoid. In addition, once learned, this process is similar to the set-up of many other Javascript frameworks.

In the beginning, unnecessary complexity can be avoided by setting up the environment in advance. While this may seem impractical, Angular templates are freely available from its website, which offers the scaffolding for a simple web app. The curriculum already accomodates this sort of gentle introduction in later programming

classes, where certain code is supplied to the students at the start of the assignment. But some activities, like Message Board, are so simple that even this might not be necessary.

To some degree, Angular obscures how Javascript and HTML function together. This is actually part of the Angular philosophy. In order to work seamlessly with the DOM, the difficult parts of the process are abstracted, letting developers spend their time on the page's functionality. Although this is a trade-off, it could be beneficial to some students. During my time working as a Teaching Assistant for CS105, I often heard students express their frustration with HTML. With Angular, it is possible to preserve these principles with less difficulty on the student's end.

As shown in Part 2 of this paper, plain Javascript and HTML do not represent the current developer's toolbox. Frameworks and libraries play a huge part in modern websites, so any exposure to these contributes to the growth of practical skills. Experience with frameworks is extremely marketable. While interviewing for BASIC, I'm certain that even my small amount of prior Javascript experience made my application more attractive. This is true for other students intending to go into software design. With the rise of full-stack Javascript, computer scientists need to consider Javascript as a language capable of solving a wide range of problems.

The introduction of more web development content will also increase interest in the department. Web development pulls from a more creative set of skills, particularly in the visual stage of website design. These projects are perfect for independent experience, which benefits anyone in the major.

V. Conclusion

Incorporating Angular into CS105 would require a complete rewrite of the course; however, it shouldn't be rejected on this basis. Instead, one alternative is the addition of a separate intro-level class teaching programming concepts specifically through the lens of web development. Another option is the creation of a special topics course designed to explore these new frameworks. Either way, this is something that needs to be addressed.

The popularity of full-stack Javascript and the number of pivotal frameworks indicates that Javascript's transformation is not short-lived. While future frameworks will eventually surpass Angular, for now it represents a facet of the field that can no longer be ignored. By learning even one additional framework, students are acquiring knowledge that will prepare them for the workplace. Once the initial leap is made to accommodate these changes, it will be easier to adapt as the field continues to evolve.

References

Borlinghaus, Tobi, and Daniel Clasen. "Comparison of 4 Popular JavaScript MV* Frameworks (part 2)." Developer Economics. March 5, 2015. Accessed November 11, 2015. http://www.developereconomics.com/comparison-4-popular-javascript-mv-frameworks-part-2/.

Hanselman, Scott. "JavaScript Is Web Assembly Language and That's OK." Scott Hanselman. May 24, 2013. Accessed November 12, 2015. http://www.hanselman.com/blog/Java ScriptIsWebAssemblyLanguageAndThatsOK.aspx.

Hevery, Miško, and Brad Green. "Keynote - NG-Conf 2014." YouTube. January 16, 2014. Accessed December 4, 2015. https://youtu.be/r1A1VR0ibIQ.

Kay, Katty, and Claire Shipman. "The Confidence Gap." The Atlantic. April 14, 2014. Accessed November 11, 2015.

Motto, Todd. "AngularJS Style Guide." GitHub. July 27, 2014. Accessed November 11, 2015. https://github.com/toddmotto/angularjs-styleguide.

Papa, John. "Angular Style Guide." GitHub. July 28, 2014. Accessed November 11, 2015. https://github.com/johnpapa/angular-styleguide.

Polepeddi, Lalith. "Made with Angular." Accessed December 4, 2015. https://www.madewithangular.com/.

Robertson, Judy. "Girls Can't Program in Their Heads" Communications of the ACM. June 29, 2012. Accessed December 4, 2015.

Wastl, Eric. "Vanilla JS." Vanilla JS. Accessed November 12, 2015. http://vanilla-js.com/.

Appendix

CS105 Rewrite:

Below is a link to code for an Angular interpretation of several assignments from CS105: Introduction to Computer Science. The activities were chosen based on the concepts each teaches. The rewrite is made up of three files. First, index.html contains the Angular directives and HTML markup. Second, ctrls.js contains the Angular code required for the page to function correctly. Finally bower.json holds basic information about the project, including project dependencies, which can be installed and managed using Bower. In order to install these dependencies, navigate to the directory containing the package file and run `bower install`, which will download the necessary source code. This rewrite relies on three dependencies: Angular, Angular-Bootstrap, and JQuery. The complete code is housed in a GitHub repository:

https://github.com/davenportn/SIP-CS105-Angular