FANTASIA: A TOOL FOR AUTOMATICALLY IDENTIFYING INCONSISTENCY IN ANGULARJS MVC APPLICATIONS.

MD RAKIB HOSSAIN
Class Roll : BSSE 0516
Exam Roll : 506
Registration No : 2012-212-073

An Academic Project
Submitted to the Bachelor of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

FANTASIA: A TOOL FOR AUTOMATICALLY IDENTIFYING INCONSISTENCY IN ANGULARJS MVC APPLICATIONS.

MD RAKIB HOSSAIN

Approved:

Signature	*Date*

Supervisor: Dr. Kazi Muheymin-Us-Sakib
Professor and Director
Institute of Information Technology
University of Dhaka

To *Hosne Ara Begum*, my mother
who has always been there for me and inspired me

Abstract

Abstract AngularJS is a JavaScript based MVC framework popular for developing Single Page Applications (SPA) development. Inconsistency may occur in AngularJS applications since JavaScript is a loosely type programing language. Most of the JavaScript errors are DOM related errors that are potentially caused by inconsistency. Mostly two types of inconsistencies namely identifier and type inconsistency occur among the modules of AngularJS. It creates hidden bugs and leads the application to perform wrong behaviors. Developers have to perform manual inspections to identify these inconsistencies.

The existing approach can detect inconsistencies only for the older version of AngularJS applications. Moreover, it also omits the presence of custom directives in those applications. The recommended angular coding style guides and the new features of AngularJS are not supported by the existing tool. It is officially recommended for the developers to follow angular coding style guides and new features of AngularJS . It is also recommended to create custom directives while developing loosely coupled modules and applications. So, while identifying the inconsistencies, in the presence of the custom directives should be considered. In order to overcome the above limitation of the existing approach, an automatic approach in the form of a tool namely FANTASIA is proposed and developed that can identify inconsistencies in AngularJS MVC applications in the presences of custom directives.

A fault injection study and comparative study are performed on fifteen AngularJS

MVC applications to check the accuracy and efficiency of FANTASIA. For the experiment, FANTASIA and existing approach ARBUSH both are implemented using JavaScript programing language. Faults are injected into the selected application according to the JavaScript MVC framework consistency models properties. Both the techniques are run on the same faulty applications, and manual inspection is also performed on those applications. It is observed that FANTASIA performs well with the 92.05% recall in twelve applications that do not contain custom directives along with developed by following angular style guides. It can also identify inconsistency the rest of the three applications with the recall of 85.6% where the presence of custom directives is considered. It is seen that AUREBESH cannot identify inconsistency in that applications that are developed by following AngularJS latest version. For conducting a comparative study, both FANTASIA and AUREBESH are run on applications that are developed by following the older version of AngularJS .The result shows that both the FANTASIA and AUREBESH perform well with the recall of 92.05% and 91.49% respectively. It is noted that recall of identifying inconsistency not changed for FANTASIA as it can identify inconsistency in both versions of AngularJS. Besides, FANTASIA can detect inconsistency in the presence of custom directives and gets 85.6% recall. On the other hand , AUREBESH cannot identify inconsistency that are present within the custom directives and gets 42.68%.

Acknowledgments

I would like to thank Dr. Kazi Muheymin-Us-Sakib for his support and guidance during the project compilation. He has been relentless in his efforts to bring the best out of me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

AngularJS is a JavaScript-based MVC framework used for developing loosely coupled web applications, which are known as Single Page Applications (SPA) [1]. This MVC framework provides developers with the flexibility to separate business logic in several reusable modules and components such as model, view, controller, directive, service, etc. However, to use these modules, developers have to follow some coding convention, for example, the view should be defined in an HTML code, and the controller and model data should be defined in JavaScript code. Developers may write the JavaScript code in the view HTML code or in a separate JavaScript file. It is recomanded for the developers to define the controller and model data in a separate JavaScript file for code readability and maintainability. The view consists of different types of DOM (Document Object Model), which is a data structure used to represent the hierarchy of HTML elements and their properties. The controller is responsible for representing the model data into the view. If any changes occur in the model data, controller function is responsible for making these change.

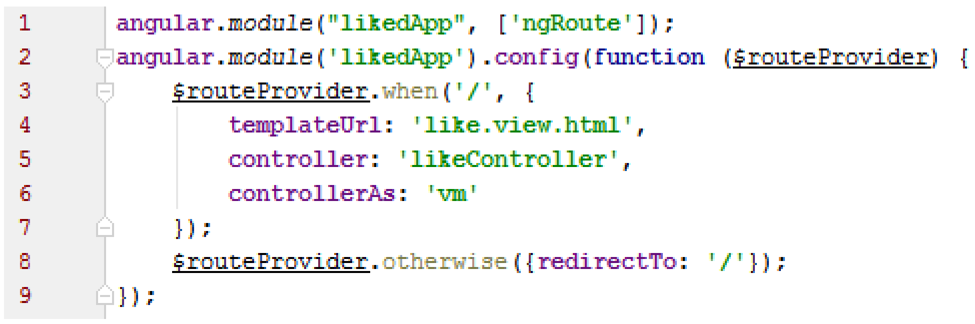
AngularJS depends on the identifier to show its functionality among the modules, especially in the view and controller. An identifier refers the name of the model variables and controller functions present in the controller [2]. To represent the functionality, the identifier of the model variable and the controller functions should be consistent in the view. Besides, in AngularJS views consist of various AngularJS built in directives such as ng-if, ng-count, that takes model variables and controller functions with a specific type [3]. So, the model variables and controller functions are used in that directive should be the consistent type. It is difficult to find inconsistencies when an application contains multiple models, views, and controllers. Developers have to perform manual inspections to find these inconsistencies. So, to identify this inconsistency problem, the aim of this research project is to develop an automatic approach in the form of a tool named FANTASIA that can identify inconsistencies in the AngularJS application. The tool performs static code analysis to identify the inconsistency. The tool can also automatically generate AngularJS applications and different modules such as a controller, view, service through scaffolding. The following sections of this chapter explain the motivation for working in inconsistency identification, the research questions of the thesis, contribution of the thesis and organization of the thesis.

1.1 Motivation

In AngularJS applications, inconsistency creates hidden bugs [4]. Sometimes developers defined some model variables and controller functions that are not used in the view. It creates redundancy in the controller and reduces the readability and maintainability of the code. To easily understand the inconsistency problem in the AngularJS applications, an example case is exhibited below.

It is mentioned earlier; AngularJS MVC applications contain several modules. The controller module is correspondent to a view module. It is recently the best practice to define the correspondent views and controllers in the application configuration file. In the application configuration file, the route of the application is defined. It describes for which view which controller is responsible.

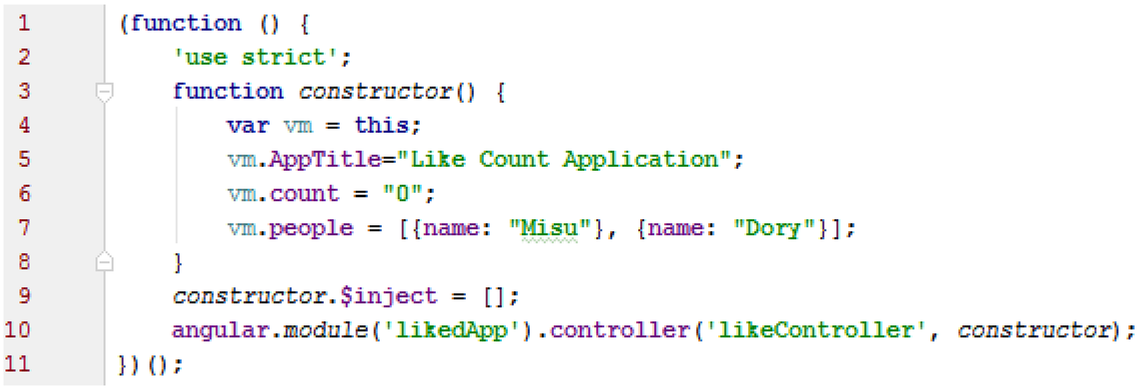
Figure1.1 shows an application configuration file. In this file, there is an app named likedApp(Figure1.1-line:1) is defined including its route configuration.



```
1  angular.module("likedApp", ['ngRoute']);
2  angular.module('likedApp').config(function ($routeProvider) {
3      $routeProvider.when('/', {
4          templateUrl: 'like.view.html',
5          controller: 'likeController',
6          controllerAs: 'vm'
7      });
8      $routeProvider.otherwise({redirectTo: '/'});
9  });
```

Figure 1.1: Application configuration file

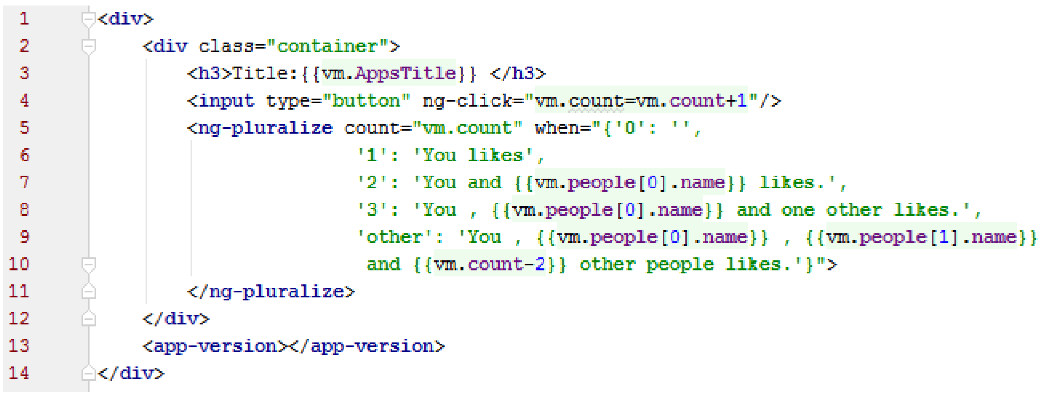
The route configuration contains many states. Here, it contains one state "/ " that refers the home path of the application. Every state also includes some properties such as templateUrl, controller, and controllerAs. The templateUrl property (Figure1.1-line: 4) refers the view of the path, and controller property (Figure1.1-line: 5) refers the correspondent controller of this view. Finally controllerAs property (Figure1.1-line: 6) refers the alice that is used to represent the controller in the view.



```
1  (function () {
2      'use strict';
3      function constructor() {
4          var vm = this;
5          vm.AppTitle="Like Count Application";
6          vm.count = "0";
7          vm.people = [{name: "Misu"}, {name: "Dory"}];
8      }
9      constructor.$inject = [];
10     angular.module('likedApp').controller('likeController', constructor);
11 })();
```

Figure 1.2: like.controller.js file

Figure 1.2 represent the controller file of the likeController. It is observed that it contains a variable named vm. It is the alice of this controller that is defined in the application configuration file (Figure1.1-line: 6).It represents the controller in the view. All the model variables and controller functions with in this controller are bind with this variable. Besides, three model variables namely vm.AppTitle, vm.count and vm.people (Figure1.2-line: 5, 6, 7). On the other hand, the correspondent view of this controller is namely like.view.html that is shown in Figure 1.3.

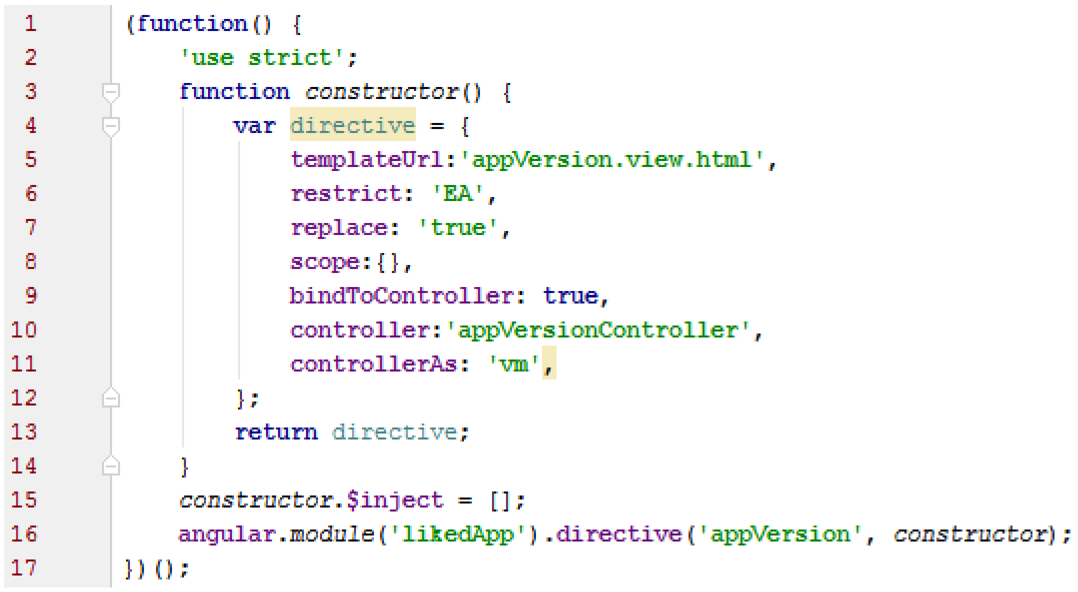


```
1  <div>
2      <div class="container">
3          <h3>Title:{{vm.AppsTitle}} </h3>
4          <input type="button" ng-click="vm.count=vm.count+1"/>
5          <ng-pluralize count="vm.count" when="{'0': '',
6                          '1': 'You likes',
7                          '2': 'You and {{vm.people[0].name}} likes.',
8                          '3': 'You , {{vm.people[0].name}} and one other likes.',
9                          'other': 'You , {{vm.people[0].name}} , {{vm.people[1].name}}
10                          and {{vm.count-2}} other people likes.'}">
11         </ng-pluralize>
12     </div>
13     <app-version></app-version>
14 </div>
```

Figure 1.3: like.view.html file

From the view, in Figure 1.3, it is seen that a model variable named vm.AppsTitle (Figure1.3-line: 3) is used here, but it is not defined in the controller as a model variable. However, the model variable defined by the controller is called vm.AppTitle (Figure1.2-line: 5) that is not consistent with its view. So, there presents an identifier inconsistency. Moreover, in the like.view.html there is an AngularJS built-in directive called ng-pluralize (Figure1.3-line: 5 to11). This directive has two attributes namely count and when (Figure1.3-line: 5). the count attribute accepts number data type. In this example, it takes a model variable named vm.count (Figure1.3-line: 5) that defined in the likeController with the same name vm.count (Figure1.2-line: 6) but with a string value rather than a number value. So, it causes a type inconsistency that leads the application to perform the wrong behavior. The application contains both the identifier and type inconsistency that cause the application not to perform its function appropriately.
It is noted that there is a HTML tag called app-version is present in the like.view.html. It is not a recognized HTML tag. It is a custom HTML tag that is created using AngularJS directive module. AngularJS allows the developer to create own HTML

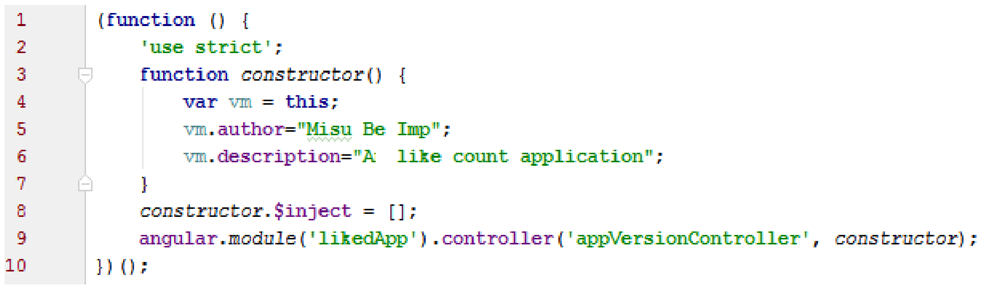
tag through using custom directives. The definition of the app-version directive is shown in Figure 1.4



```
1   (function() {
2       'use strict';
3       function constructor() {
4           var directive = {
5               templateUrl:'appVersion.view.html',
6               restrict: 'EA',
7               replace: 'true',
8               scope:{},
9               bindToController: true,
10              controller:'appVersionController',
11              controllerAs: 'vm',
12          };
13          return directive;
14      }
15      constructor.$inject = [];
16      angular.module('likedApp').directive('appVersion', constructor);
17  })();
```

Figure 1.4: appVersion.directive.js

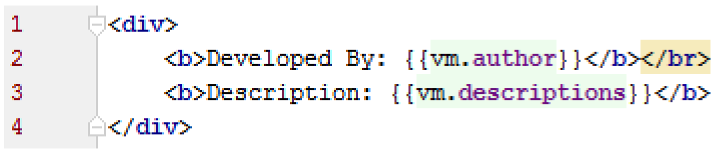
It is seen from the Figure 1.4 that the app-version directive contains various properties such as templateUrl, restrict, controller, controllerAs etc. It may also contain view and controller depending on the type of this directive. The templateUrl (Figure1.4-line: 5) property represents the view file of this directive. The controller and controllerAs represent the controller and the alice of this directive controller. The details about directive are found in the chapter 2 section X. Generally, the view displays the output of the directive, and the controller contains the data that is used by the directive. Figure 1.5 shows the appVersionController of the directive app-version (Figure1.4-line: 16).



```
(function () {
    'use strict';
    function constructor() {
        var vm = this;
        vm.author="Misu Be Imp";
        vm.description="A  like count application";
    }
    constructor.$inject = [];
    angular.module('likedApp').controller('appVersionController', constructor);
})();
```

Figure 1.5: appVersion.controller.js

The controller contains two model variables namely vm.author and vm.description (Figure1.5-line: 5, 6) the correspondent view of this controller named appVersion.view.html is shown in Figure 1.6. It is observed that the value vm.descriptions used in the view (Figure1.6-line: 3), is not defined in the controller.The controller contains a model variable named vm.description that is not used in the directives view. So, it creates identifier inconsistency with in the app-version directive.



```
1  <div>
2      <b>Developed By: {{vm.author}}</b></br>
3      <b>Description: {{vm.descriptions}}</b>
4  </div>
```

Figure 1.6: appVersion.view.html file

Researchers proposed various approaches to identify inconsistency. For example Michael et al. (2015) [5]. proposed an approach named Type-Devil that can detect type inconsistency in the dynamic programing languages such as JavaScript. However, it cannot detect inconsistencies in AngularJS MVC applications. The reason it that inconsistencies occur in the AngularJS applications between the HTML code and JavaScript code. However, Type-Devil can only find an inconsistency that only presents in the JavaScript code. Besides, to detect the inconsistency in the JavaScript applications, an automatic approach is proposed by Frolin et al. (2015) [6] in the form of a tool named AURBESH. It can detect the identifier and type inconsistency in the AngularJS applications. However, this tool is not compatible with the new AngularJS features and fail to detect inconsistencies in that application that is developed by following AngularJS style guides. Moreover, this approach cannot detect inconsistencies that present in the custom directives since it omits the present of custom directives in the applications.

1.2 Research Question

In order to accurately identify inconsistency among the associated modules, MVC groping should be built in the presence of custom directives. Existing tool AURBESH may find inconsistency in AngularJS application ignoring the presence of custom directives. Moreover, it may only work for the older version of AngularJS 1.x application. So, it fails to detect inconsistency from the code, which is written by following the recent coding style guide, without considering the application structure. It is inferred that existing tool may not accurately detect inconsistency in AngularJS 1.x MVC applications. So, this is lead to the following research question .

How to accurately identify inconsistency in AngularJS 1.x MVC application?

This question is associated with some sub-questions. So it will be answered by the following sub questions.

1. How to build MVC grouping in the presence of custom directives from different application structures? To answer this sub-question following steps will be adopted
   (a) A generalized application structure should be defined to locate the files, associated with the application.
   (b) Every file name should be descriptive enough to be able to figure out which section or component it is in and what type of AngularJS object it is [7].
2. How to identify inconsistency in the presence of the latest version of AngularJS 1.x MVC application considering recent coding style? The following steps are followed to answer this sub-question.

(a) Application should be built by following the recommended syntax and coding style using angular style guide over any other syntax.

(b) If any application contains primitive coding syntax and AngularJS version, it will be re-factored to recent AngularJS version by following the angular style guide.

1.3 Contribution of This Research Project

In this research based academic project, a technique is developed namely FANTASIA in the form of a tool that can identify inconsistency in AngularJS 1.x MVC applications. Static code analysis is performed by this tool to find the inconsistencies. Initially, the application configuration file is extracted to find the associated views, controllers and alices. All the related view and controller files are extracted. This extraction provides the model variables and controller functions that are defined in the controllers and used by the views. After that, it is checked if there any custom directive presents in the view. If the custom directive is used by the view, the directive and its associated files are also extracted. It provides the model variables and controller functions that are related to the directive. Finally, a list of the mvc group is built by using the associated model, view, controller and directives. Eventually, the consistencies are checked for each of the element of the group to identify the inconsistencies.

To evaluate the accuracy and performance of this technique and the tool, a fault injection study and a comparative study in performed. The tool FANTASIA and the existing tool AUREBESH are developed using JavaScript programing language on top of Node.js framework. The studies are performed on fifteen AngularJS MVC applications as an experimental dataset. The dataset is divided into two categories namely presence of custom directive and absence of custom directive. Among the fifteen applications, twelve applications do not contain custom directives. The rest of the three applications contain custom directives. The outcome of the fault injection study is that the tool FANTASIA gets 92.05% recall for identifying inconsistencies in those twelve applications that do not contain custom directive. It also acquires 85.6% recall for identifying inconsistencies in present of custom directives.

1.4 Organization of The Document

This section provides an overview about the remaining chapters of this thesis. The chapters are organized as follows.

Chapter 2: JavaScript, Framework, JavaScript MVC Frameworks, AngularJS different types of AngularJS modules are exemplified in this chapter. Besides, inconsistency issues and importance of inconsistency identification are also discussed.

Chapter 3: This chapter deals with the existing works in literature regarding the JavaScript MVC Frameworks, AngularJS and inconsistency issues in JavaScript.

Chapter 4: The proposed approach namely FANTASIA is explained in this chapter. All required procedures, modules and algorithms are also mentioned in the following sections of this chapter.

Chapter 5: The implementation of FANTASIA and existing tool AUREBESH are explained in this chapter. A fault injection study and a comparative result analysis between FANTASIA and AUREBESH are also discussed in this chapter

Chapter 6: It is the last chapter that contains a discussion about the proposed approach, important threats to validity and some future directions.

Chapter 2

Background Study

This chapter comprises the initial terms and terminology associated with this research project. It helps to understand the research work properly. First few sections includes JavaScript and its importance, concept of MVC patterns, web application framework and JavaScript MVC framework. Since the research work is based on AngularJS, the rest of the sections contain more details about AngularJS framework. Those section comprises introduction to AngularJS, modules and features of AngularJS, AngularJS style guides and best practices, inconsistency issues in AngularJS applications and application of inconsistency identification.

2.1 JavaScript

To develop both client-end and server-end application, JavaScript has become the most popular programming language for the developers. It is a dynamic interpreted programming language that almost follows the syntax of C programming languages. It gives developers the flexibility to write code without following any strict typing discipline. It contains some important features that are increasing its popularity. Some of the most favorite features are, creating dynamic objects that can be modified at run time, the literal notation which is used to declare objects and arrays by listing their components [3] and loose typing variables, functions and object declarations. However one of the most important features is first-class functions that allow developers to pass a functional expression as parameters or return functional expression as values. JavaScript is object oriented programming languages that allow an object to get some properties from another object by following prototypical inheritance. It also allows the developers to follow the functional programming style to do functional programming.

Initially, JavaScript was designed as a scripting programming languages. It is embedded in a web page with few codes to extend functionalities. With the increasing number of devices, JavaScript is now being used developing both the mobile application and native desktop application development. For these different types of applications, better user interfaces and experiences are needed. Technology that contributes to the improvement of better user experiences is called AJAX. It allows to communicate and interchange the data between the server-end and client-end asynchronously without interfering the display and the functionality of the web pages

2.2 The Model-View-Controller Pattern

The Model-View-Controller (MVC) is known as an architectural pattern that was first introduced by Trygve Reenskaug in 1978 [8]. It was considered to be a solution that helps user's mental model .By associating with the domain of the application it enables them to inspect and modify their information [7]. That time the MVC pattern was used as an approach to representing information, display and control for the set of reusable system components used by programming environment [9]. MVC pattern allows developers to structure applications that encourage the separations of reusable components to represent

- The application domain data in the form of the model.
- The way to serve application data to the user.

- The way how a user can interact with the domain data model.

There are three major components of MVC architectural patterns. The components are the model, view, and controller. The detail description of those three components is presented below.

1. **The Model**: Model is the representation of application domain or knowledge. It is the part of the application that contains the actual data. It may contain a single object or a structure of complex objects. The model also contains logics such as validations and access control. The model is the actual heart of an application, and without it, the application will not work [10].

2. **The View**: A view is the visual representation of its model data. It generally focuses the certain attributes of the model data and suppresses others. Thus, it is treated as a presentation filter. It is attached to its model to get necessary data for the external representation by asking questions. It may also update the model data by sending appropriate messages. The questions and messages have to be in the terminology of the model. Therefore, the view will have to know the semantics of the model attributes [10].

3. **The Controller**: A controller is a link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages, and pass these messages to one or more of the views [10].

2.3 Framework

A software framework is a set of source code and libraries that provide help to the developer when coding. A framework can help with a lot of different aspects of an application such as data storage, security and user interfaces [11]. For example, a security framework can be used for an application. The developer will not need to implement the authentication and authorization functionality since it is already implemented and provided by the framework. The developer will only need to use the supplied component that has the required functionality, and may make smaller changes to fit the application.

A web application framework is a type of a software framework that helps developers specifically when developing web applications [12]. There are many different frameworks available that all are designed to fulfill some requirements. Frameworks can be applied to many areas such as education or health care, but in this research, the focus is on software engineering and more specifically web programming

2.4 JavaScript MVC Frameworks

To build a small and light website or applications on the client end, plain JavaScript is used for most recurrent requirements such as AJAX communications, DOM manipulation, etc. However, web applications are developed to manage a large quantity of data with a better user interaction and interface. Therefore it is necessary to use and write a significant amount of code for satisfying those requirements. To deal with low-level API, abstract interaction with DOM, and complex way to use AJAX calling, Jquery [13] was created to provide a better solution. Even, when developing large web applications with complex user interaction, it is not expected to load the full page for every page request. It can be done using the DOM manipulation libraries. However, the DOM manipulation libraries cause a big amount of nested and unmaintainable code without any structure. As a result, JavaScript MVC framework was emerged to provide a basic way of organizing the applications code in a maintainable way [14].

JavaScript MVC frameworks are proposed to acclimate the MVC architectural pattern to provide an abstraction layer which simplifies the implementation of separation of concerns. The separation of concerns refers to the code for each of these concerns separate. Changing the interface should not require changing the business logic code, and vice versa [15]. At present, a large numbers of JavaScript MVC frameworks is created for fulfilling applications requirement in various ways. The top most used JavaScript MVC frameworks are AngularJS, BackboneJS, and EmberJS [16]. These frameworks are similar to each other and the concept of using models, views, and controllers. The oldest framework is EmberJS which is developed in 2007. The recent frameworks are AngularJS and BackboneJS developed in 2009 and 2010 respectively [2]. These are represented in on-line to open source community. The community provides help to each other and finds problem solution.

2.5 Modules and Features of AngularJS

AngularJS is a client-end web application framework famous for developing Single Page web Application (SPA) [1] development based on JavaScript programming language. It is developed and maintained by Google. It follows the MVC architectural frameworks for developing web applications. It is an open source and completely free for developers to use. AngularJS is licensed under the Apache License Version 2.0 [2].

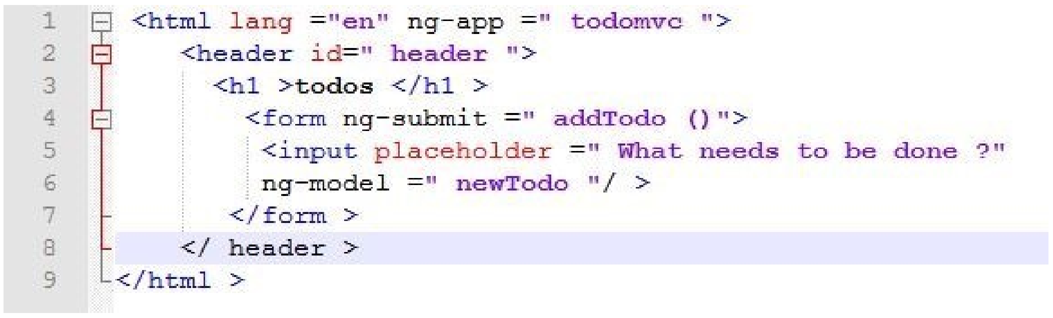
1. **Modules**: An AngularJS application is developed by using a set of modules. The modules are considered as a container to the various part of the application. Those are created as a reusable component of the application

[17]. Modules may depend on other modules. Modules may take another module as its dependencies. It is defined by using the methods of modules. Those methods are defined by the AngularJS.

2. **Services**: An AngularJS service is an object which allows encapsulating the functionalities associated with a specific concern. It follows some design pattern such as singleton, factory, and constructor. Singleton is meant that an object has only one instance at a time. Service is instantiated as a singleton by using factory or constructor function provided by AngularJS. All the different components that depend on services may share the created singleton, for example, controllers, directives, filters and other services. To use a service by other components of the application, the name of the service must be specified inside that components and AngularJS inject the service into that components for its usage. These services are stateless objects holding a set of methods to interact with various aspect of the applications for example server request, manipulations of arrays, asynchronous operations, etc. The separation of concern is the primary purpose of the services. There are few built-in services provided by AngularJS to deal with the common interest in the development of applications such as $http, $filter, and $timeout [17].

3. **Template**: In a web application, HTML is used to define the structure of the document to the user. The HTML documents are parsed by the browser for generating Document Object Model (DOM). It models the actual document presented to the user. For modifying, styling the document, the DOM can easily be accessed dynamically through program and scripts. AngularJS uses this fact for supporting the DOM-based templates [17]. These templates are based on HTML containing the necessary elements and attributes to render dynamic interface which a user sees in the browser window. Figure 2.1 shows a sample template. It includes the definition of the document (HTML

element in line 1) and the markup for the main input of the application, which is represented by the form element (line 5).



```
1 <html lang ="en" ng-app =" todomvc ">
2     <header id=" header ">
3       <h1 >todos </h1 >
4         <form ng-submit =" addTodo ()">
5           <input placeholder =" What needs to be done ?"
6           ng-model =" newTodo "/ >
7         </form >
8     </ header >
9 </html >
```

Figure 2.1: Angular Appication

4. **Directives**: A specific HTML markers used in the template to define the UI behavior is known as a directive. Directives are executed by AngularJS and have the ability to register event handlers, modify the DOM structure. Those can be found in the templates taking the form of HTML constructs such as element tags, attributes, CSS classes and event comments. While compiling the directives, AngularJS traverse the DOM generated by the browser and searches for all directives. These directives are executed in order of priority creating the final DOM presented to the user. Directives can (1) use the same scope of the parent element; (2) create a scope that inherits from the scope of the parent element, or (3) create an entirely new scope. It is also possible to create custom directives. By default, a directive utilizes the parent scope without creating the new scope.However, in many cases, this is not what [18]it is wanted. If directive manipulates the parent scope properties slowly by creating new ones, that pollutes parent scope. It is not a good idea to let all the directives use the same parent scope.The reason is that anybody can modify scope properties. So, the following guidelines may help developers choose the right scope for building custom directive.

   - **Parent Scope**: This is the default case. If directive does not manipulate the parent scope properties, there is no need to create a new scope.

In this situation, using the parent scope is considered to be the best practice [18].

- **Child Scope**: This creates a new child scope for a directive which prototypically inherits from the parent scope. If the properties and functions set on the scope are not relevant to other directives and the parent, a new child scope should be created. With this, all the scope properties and functions defined by the parent is available in child scope [18].
- **Isolated Scope**: It is needed if the directive is self-contained and reusable.The directive may create many scope properties and functions that are always required for internal use .Besides, it should never be seen by the other components of the application. If this is the case, it is the best practice to create an isolated scope. It is expected that the isolated scope does not inherit the parent scope [18].

In Figure :2.1 the ng-app attribute (line -1) is an AngularJS directive that defines the root node of the application.

5. **Expression**: AngularJS expressions are expressed by double curly brackets (expression) or are the values of some directive attributes. Literals such as arrays ([]), objects (), operators and variables are examples of elements which can be used in expressions. Expressions are evaluated using a context represented by an object, called scope. Variables and functions used in expressions must be defined in the scope object. During the template compilation, when the ng-app directive is parsed, AngularJS creates an object representing the main application scope, which is referenced as the rootScope.Since directives can define different scopes from the rootScope, expressions from various parts of the template may be evaluated under different scopes. When the value of an expression changes, AngularJS updates the view accordingly [17].

6. **Controllers**: In AngularJS, controllers are used to managing the state of applications that related to the presentations. It also provides necessary interfaces to modify the presentations. Usually, controllers are used with the ng-controller directives provided by the AngularJS. When controller directive is used inside a template, it takes the name of the controller. Besides a new scope created with this controller directive that passes as a parameter of specified controller. The responsibilities of the controller functions are to populate the new scope object with properties and methods which will be available in the template for the evaluation of expressions [17].

7. **Digest Cycle**: AngularJS consistently maintains in sync the state of the application with the view presented to the final user. The framework provides this synchronization by comparing the current value of all variables in the scope referenced by the template expressions with their previous values. When a change is detected, the framework adequately updates the DOM, in a process known as digest cycle [17].

2.6 AngularJS Style Guides and Best Practice

Style Guide: A style guide (or manual of style) is a set of standards for the writing and design of documents, either for general use or a specific publication, organization, or field. A style guide establishes and enforces style to improve communication. To do that, it ensures consistency within a document and across multiple documents and enforces best practice in usage and language composition, visual composition, orthography and typography [19]. Google provides an official AngularJS style guide and the best practices, but the most accessible and comprehensive guides are from the AngularJS community [19].

- Minko Gechevs AngularJS Style Guide

- Todd Mottos AngularJS Style Guide
- John Papas AngularJS Style Guide

It is hard to say that which is the best style guide .The reason is that they are all good style guides. John Papas guide is comprehensive and evolving, Todd Mottos is concise and excellent to start, and Minko Gechevs is translated in different languages. However, it seems that John Papas style guide has been recommended officially by Google by the most current and detailed AngularJS Style Guide. The following principles [20] are the most important AngularJS-specific points in the Jonh Papas style guide - **The LIFT Principle**

1. Locate the code quickly
2. Identify the code at a glance
3. Keep the Flattest Structure
4. Try to stay DRY (Dont Repeat Yourself)

2.7 Inconsistency Issues in AngularJS

AngularJS is susceptible to consistency issues for the abstract interaction among the modules. To represent functionality of an application, it depends on the use of identifiers. So, the definition and use of these identifiers are required to be consistent across associated modules. Since JavaScript is loosely typed dynamic programming language, the developers have to keep in mind that, values assigned to the model data and returned by the controller functions, should be consistent with their expected types. Inconsistencies among these identifiers and the types can potentially incur a significant loss in the functionality and performance. The reason is that the major features of an application rely on the controller functions and the model data. Moreover, developers declared some model variables and controller functions though these are not used in the view. It makes the code smelly with redundancy. It also reduces the readability and understandability of the code. Besides AngularJS supports the DRY (Dont Repeat Yourself) feature. It refers, just creating one directive and using it anywhere within the entire application. Despite having a lot of built-in directives, it also enables the developer to create custom directives to improve the understandability and readability of the code. Every custom directive has some specific properties that define its template view, model, and controller. Sometimes, it is also used inside a view under a particular controller by following parent-child relationship. While using custom directives, inconsistency may arise not only within its own, model, view and controller but also between its parent view and controller

2.8 Application of Inconsistency Identification

It has been observed in some studies that almost 65% of the faults and bugs in JavaScript are related with DOM API methods call [4]. The DOM API is the origin of inconsistency due to the wrong interaction between DOM object and JavaScript code. This inconsistency is acute in the presences of a JavaScript DOM manipulation libraries. The reason is that it abstracts the DOM API method call between the web pages and JavaScript codes.

AngularJS framework depends on identifiers of model variables and controller functions to represent the functionality of the modules such as model views and controllers. Inconsistency among those identifiers may cause hidden bugs in the applications. Since these modules communicate with each other via data, any data type inconsistency may result in unexpected behaviors in the application without showing any error [4]. Besides, it has been observed that sometimes developers expose some model variables and controller functions that have never been used in the view. It not only creates redundancy but also increases the line of code and impairs the code understandability. It is hard to detect identifier and data type inconsistency along with redundancy when an application contains multiple models, views, controllers and custom directives. Developers have to perform manual inspections to find this inconsistency. Identifying inconsistencies in AngularJS MVC applications is really needed to reduce application faults. This research will assist developers to accelerate their application development. The reason is that it helps the developer to automatically detect inconsistency without manual inspection.

2.9 Summary

A brief description of JavaScript, MVC frameworks, has been discussed in this chapter. Besides, AngularJS, a various module of AngularJS are also described precisely. Inconsistency issues in AngularJS applications, the effect of inconsistency in the application development are mentioned in this chapter. Existing approach and other related works on AngularJS are discussed in the next chapter.

Chapter 3

Literature Review

AngularJS has gradually been developed for the last couple of years; significant work has not been found on AngularJS. Moreover, no early work has been found that discusses consistency issues in AngularJS applications to the best of author knowledge. However, few works address MVC pattern appliance in the clientend application development. Besides, few more works are found that rigorously discuss the DOM-related faults and errors. By analyzing those works, it is observed that most of the JavaScripts faults are DOM related. Moreover, and the primary reason of the DOM-related faults is an inconsistency between the JavaScript code and the DOM element. Two works are found that discusses inconsistency issues in JavaScript application. Several survey studies are also conducted on AngularJS to find the standard errors and best practices that are followed by the developers. Based on all of those works knowledge domain of this research work is categorized in four section

- Application of MVC Pattern
- DOM-related faults in JavaScript.
- Inconsistency in JavaScript.
- Survey Study on AngularJS.

This chapter contains the use of MVC pattern for developing client-end applications is discussed. It contains the appliance of MVC pattern for developing fixable web application, and meme media technologies. The next section deals with the DOM-related errors and faults in JavaScript application. An empirical study along with different types of approaches are also discussed in that section. The following section contains the inconsistency issues arise in JavaScript application. The last section comprises with survey studies conducted on AngularJS.

3.1 Application of MVC Pattern

MVC is an architectural pattern previously that has been applied to the server end of a web application. The controller and the model are implemented on the server-end, and the view is represented by the output of the application to the client-end. Besides, the use of MVC pattern in client-end has been recently introduced, so there is only a few papers and works addressing the topic. Most of these works discuss the application of MVC pattern on JavaScript application. Avraham Leff et al (2001) introduced Flexible Web-Application Partitioning where developers may apply MVC pattern in a partition-independent manner [21], Behnam et al (2011) introduced a very simple design pattern for Rich Internet Application (RIA) based on MVC design architecture, [22] and Jun Fujima [23] et al. suggested a prototypical implementation of a meme media platform with a modern JavaScript framework [23]. Those works are elaborately discussed below below

3.1.1 Flexible Web Application Partitioning

MVC architectural design pattern is an effective way to build interactive applications. It is also known as Presentation Abstraction Control (PAC) design pattern [24]. It decouples the applications business logic from the user interface. The Presentation is comprised with MVCs View and Controller and the application's data known as Model is termed as Abstraction. The Control component does the communication between the Presentation and Abstraction component.

The use of both PAC and MVC design pattern makes it flexible to build and maintain applications. The reason is that the applications look and feel can be changed without changing the business logic and applications data. However, there are problems to use MVC design pattern for developing web applications as web applications are intrinsically partitioned between the client-end and server-end. Three types of deployment architecture are followed such as thin-client, fat-client and dual-mvc. When the View is provided from client-end and the Model and Controller are provided from the server, this approach is called thin-client. In fat-client approach, the Model, Controller, and View are also provided from client-end. For dual-MVC approach, both the Controller, Model reside on both the client-end and server-end.

Avraham Leff et al. (2001) [21] introduced Flexible Web-Application Partitioning where developers may apply MVC pattern in a partition independent manner. By flowing the Flexible Web Application Partitioning, the application can be developed and tested in a single address space along with deployed various client-server architecture without changing the applications. It also helps to take and change partitioning decisions without modifying the applications.

3.1.2 Meme Media Technology

In recent web applications, World Wide Web (WWW) is a potential platform. With the separation of the next generation of HTML, known as HTML5 gets the popularity of the World Wide Web Consortium (W3C). Therefore, for building a rich web application, separated web-based state of the art technologies becomes the first choice of the developers. With those modern technologies, developers can implement rich web applications. The purpose of the developing Meme media [25] technology is to provoke and support the evolution of the knowledge and technol-

ogy and the Internet. It provides media objects which were known as memes first introduced by Dawkins [23]. It allows users to modify and redistribute different knowledge and resources wrapped by media objects. The user can combine media objects through manipulation of those for example drag and drop or copy and paste to compose new objects without any coding. By using this features, when one user may upload a new media object on the computer network another user may download it and reuse it. Moreover, another user may combine the object with another object to compose new object to publish it to the network. With this cyclic process of the user interaction with these meme media objects, various types of meme media objects are accumulated on the network.
Jun Fujima et al. [23] explore the possibility of meme media platform with simple Web technologies that are available to more different types of devices. The implementation of client-side media object and functionalities such essential media object architecture and combine the media objects are focused on this work. In the recent JavaScript development, there are some problems with the compatibility among different browsers and over flexibility. To avoid the complexity researcher chose a base JavaScript framework AngularJS for developing a structured web application. The researcher using AngularJS developed a prototypical implementation of meme media platform. Also, several media objects are prepared on a different platform and tested operation of them on multiple Web browsers. The reason is that they used only pure HTML and pure JavaScript to implement the platform .Moreover, they develop the platform that works on modern Web browsers that support latest HTML5 and JavaScript specifications.

3.2 DOM Related Faults in JavaScript

The recent web application gets information asynchronously from the server-end without reloading the whole web page. This approach is much more user interactives from than traditional web applications. By the use of JavaScript at the client -end this interactivity is accomplished. JavaScript allows the creation, modification and delectation of the node of the tree data structure namely Document Object Model (DOM).DOM is a data structure like a dynamic tree that represents the hierarchy of HTML elements and their properties in the web pages. Almost 97 of the top 100 most visited websites [26] and web applications use JavaScript at client-end for better user interaction. JavaScript is a weakly typed programming language that supports execution of new code at run time. It is observed that this factor leads to many programming errors. Moreover, the web browser is enduring to errors in JavaScript code, though those are different in the event of error handling. When an exception occurs in JavaScript code, web browsers do not stop executing rather those continue executing in response to user events and notifications. For this reason, it is difficult to find JavaScript bugs and errors during testing. bugs and errors during testing

3.2.1 JavaScript Errors in the Wild

In a research work, Frolin et al. (2011) [27] tried to conduct an empirical study based on an error in JavaScript based web applications and categorize the standard errors in these applications. The sources of these errors were also analyzed by performing the dynamic and static analysis. Another objective of this research was to provide guidelines and principle that helps developers and testers enhance their web applications readability. In a JavaScript based web application, an error may have different results with a loss of its functionalities. It is necessary to study the readability of the JavaScript code to understand the errors occurring in the web applications. The study was based on the error message printed to the JavaScript console while executing the application in the browser. Firebug was used to capture the message. The error messages were analyzed and categorized and also correlated with the web applications with the types and frequencies of the error message to understand their relationship. The outcome of the work was, a systematic approach to executing web applications in different testing mode and categorize the error messages. The method was implemented as a tool based on the existing approach. The tool was also run on 50 of the 100 most visited websites to study and characterize the errors. It was found that almost 93% of the errors fall into one of four categories: Permission Denied (52%), Undefined Symbol (28%), Null Exception (9%), and Syntax Errors (4%).

3.2.2 AutoFlox

Testing is the most generic way of acquiring confidence in software reliability. On the other hand debugging of the web application is considered to be most expensive and manual task. However, locating the faults or fault localization is still the most expensive among other activities of the debugging process. The fault localization process starts when developers observe an error in the web applications in an automated testing or manual inspections. After that developers try to understand the cause and the nature of this errors . It is done by reviewing the JavaScript code and examining the DOM tree. After getting the fault, developers modify the code, run the application and go through the navigational actions the lead to the wrong state or execute the test case. It is hard and time-consuming for the developers to inspect the JavaScript errors manually. The reason is that JavaScript is loose error detection semantics and loosely type with dynamic nature. Therefore, an error may remain undetected in the application for a long time before triggering an exception. Besides errors may arise because of the asynchronous and dynamic interaction between the JavaScript code and DOM tree that makes it difficult to understand the cause of the error. Finally, errors may occur while using third party code such as libraries, widgets, etc.

Ocariza et al. (2012) [28]conducted a study over 29 openly available JavaScript bug reports from four open source web applications such as TUDU contains 9 bugs, TASKFREAK contains 10 bugs; WORDPRESS contains 11 bugs, and Google contains 12 bugs. The key challenges to perform the study was to get the available bugs. There are very few web applications publish their bug database. After analyzing, the bugs were categorized into five types such as

- Code-terminating DOM-related.
- Output DOM-related.
- DOM-related error of unknown kind.
- Non-DOM-related error.
- Output DOM-related.
- Unknown JAVASCRIPT error.

An automated technique was proposed that lessen the difficulties with manual web fault localization. The techniques were developed on dynamic backward slicing of the web application to localize the JavaScript faults. The fault localization technique was implemented in a tool namely AUTOFLOX. It is evaluated on three open source web applications and a production application. The outcomes of this work are given below

- Identifying DOM-related JavaScript faults are considered to be a new problem space.
- For locating the DOM-related errors in JavaScript code, an automated technique is developed.
- AUTOFLOX, an open source tool that implements the proposed fault localization technique.
- An empirical study that examines the proposed approach and its efficiency in real world relevance. The study indicates that DOM related errors from

the majority of reported JavaScript errors. About 79% of the reported JavaScript errors are the DOM related.

3.2.3 Empirical Study

Most of the time JavaScript code is written to interact with the DOM. JavaScript can access the DOM and modify the DOM dynamically using DOM API methods that may change the content of web pages without reloading the whole page. Though these features make the web application highly interactive, it causes additional faults in JavaScript code.
A study is conducted by Karthik et al. (2013) [4]to collected console message from 50 popular web application to understand and explain how JavaScript faults occur and what kinds of faults appear in the web applications. It is found that on average four JavaScript console message appear in these modern web applications, and these messages were falling in five different categories. However, that study did not explore the cause and did not analyze the types of faults. It is important to the developers and testers to understand the reason using a dynamic analysis tool to increase the reliability of web applications.

An empirical study was performed on 300 JavaScript bug reports .The researchers were interested to discover the cause of JavaScript faults .They analyzed these to find their consequences. The bug reports were chosen on the basis of how the applications behave when faults occur.So, the main challenges are to study the bug reports. However, 317 bug reports are collected from 8 web applications and 4 JavaScript libraries. After conducting the study, researcher categorized the faults in five different classes. It is found that most of the faults were related to DOM-related faults. The causes and effects of these faults are analyzed. Their result showed that almost 65% of JavaScript faults are DOM-related faults that occur because of the wrong interaction of JavaScript code and DOM element using incorrect identifier. Moreover, it was found that DOM-related faults are responsible for 80% of the highest impact faults in the web application. It was also observed that majority of faults occurs due to JavaScript code instead of server-end code. Besides, few coding patterns lead to raising these faults.

3.2.4 Vejovis

JavaScript applications are proms to errors because of its dynamic nature. An empirical study [29] shows that around 65% JavaScript faults are associated with DOM-related faults that occur because of the wrong interaction of JavaScript code and DOM element. Besides it is found in the study that about 80% of the highest impact of JavaScript faults are also DOM-related faults. Moreover, it is observed that DOM-related faults require more time to fix.

In this research work, the primary goal of the researcher was to propose an approach that facilitates the process of fixing the DOM related JavaScript faults by providing the recommendations during web application testing and debugging task. First, it was observed on many DOM-related faults to find the typical pattern on how developers fix the DOM related bug during application developments. After that, an approach was proposed that automatically fixes the DOM-related faults. The approach included the combination of both static and dynamic analysis to identify the lines of codes through backward program slicing for getting the assignment values of DOM elements. After getting the lines, a string solver was used to find the candidate replacement DOM elements and propagate the values along with backward slice to find the fix. The approach was implemented as an open source tool named VEJOVIS. It was deployed on a web application after the occurrence and subsequent localization of JavaScript faults. Some common types of DOM-related faults were categorized by the researchers based on 190 bug reports. It was found that modifications of DOM method or property or assignments values into valid replacements values are the most common faults.
All those works are directly related to DOM related errors and faults. However,

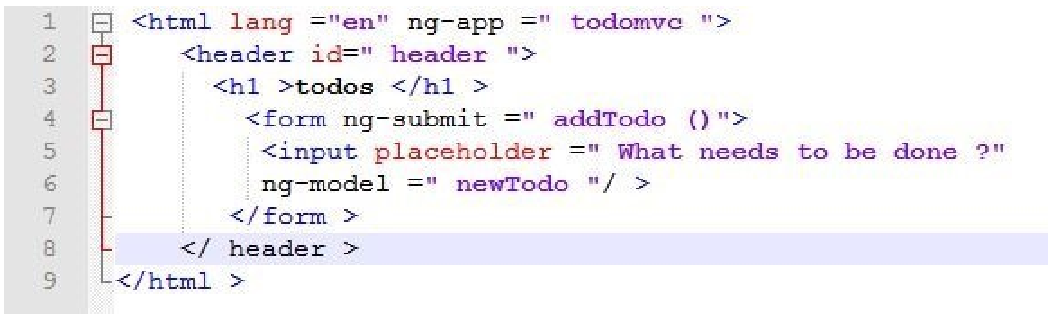
they are only capable of finding DOM-related faults in native JavaScript web application. These approaches are not capable of finding inconsistencies in AngularJS applications since they omit the presence of the JavaScript MVC framework applications.

3.3 Inconsistency Issues in JavaScript

JavaScript makes developers not to annotate their programs with type information or to follow any strict typing discipline. This practice helps the developer to write many lines of code within a short period. Although these are beneficial features of dynamic programming, the freedom provided by dynamic programming languages often causes some hidden bugs. Since the dynamic programming language does not impose developers for any typing discipline. No compile-time warnings are reported if a program has data type's inconsistency. Even sometimes, many dynamic languages silently compel values from one type into another type that leads to the incorrect application behavior without showing any obvious sign of errors.

3.3.1 Type-Devil

Michael et al. (2015) [5] deal with a data inconsistency problem of dynamic programming languages. For easily understanding this problem, an example is provided with a small code block along with a clear explanation. Figure 3.1 shows a block of code. Two functions are described namely addWrapped(x, y) and Wrapper(v). It is observed that addWrapped(x, y) function takes two parameters (e.g. x, y) as objects with a property v. If the second parameter namely y exists, the function returns the sum of the property (v) of those two parameter objects. If only one parameter exists, then the function returns only the parameter's object property namely v. Another function named Wrapper (v), wrap a number property v within an object.In JavaScript, this type of function is called Constructor function that instantiates an object. There are two ways for instantiating these parameters as objects.Developers may instantiate objects using the Wrapper () constructor function (e.g. new Wrapper (5)), or using the JavaScript object notation such as v: 5.



```
<html lang ="en" ng-app =" todomvc ">
    <header id=" header ">
      <h1 >todos </h1 >
        <form ng-submit =" addTodo ()">
          <input placeholder =" What needs to be done ?"
          ng-model =" newTodo "/ >
        </form >
    </ header >
</html >
```

Figure 3.1: JavaScript type inconsistency example

In line-10, the function is called, and it returns only 23 as it takes one parameter. So it returns the object's property with a name v. In line-11, the constructor function is also called, as expected it also returns 23 as it takes two parameters by instantiating two parameters objects with two different approaches. However, when the function is called in line 12, it takes two parameters where the property of first parameter object is a string instead of a number. So while calling the function, it concatenates the properties of those two parameters object and returns 185 instead of 23.

For identifying these types of inconsistency issues for dynamic languages researchers propose an approach named TypeDevil as a tool that detects such problems by dynamically analyzing the program and by reporting variables, properties, and functions that have inconsistent types

The approach has three steps; it starts with gathering the Type observation. In JavaScript, data types are considered structural types. On the other hand, the approach represents data types as a record of typed properties. Here Type is either a primitive type (Boolean, Number, String, Undefined, or Null) or a record type that maps named properties to sets of types. A record type represents one of the four types object type, array type, function type, and frame type, where properties represent local variables of a function. Two types are the consistent type if both types are the same, if both types are structurally equivalent, or if one type is a structural subtype of the other type. There is a dynamic analysis of source code that gathers the observation of each variable, properties and functions. It instruments the program by adding code that records the type of each reference. This approach adds code in different instrumentation points of the source codes. The researcher mentions six instrumentation points for adding code such as object literals, get property and put property, function literals, function calls, function enter and exit, and variable reads and writes.

After gathering type observations at runtime, all observations are merged into a type graph. The type graph of an execution of a program is a directed graph. The nodes represent types observed during the execution. An edge represents that the property p of type t1 that has been observed to point to a value of type t2.The final step of this approach is to report type inconsistencies. Initially, the analysis considers each property of a type as inconsistent where the type node has more than one outgoing edge labeled with this property. For evaluating the effectiveness of TypeDevil, they apply the approach to real-world web applications. It effectively finds inconsistent types, many of which correspond to problems that programmers should be aware. In total, the analysis reports 33 warnings, of which at least 15 correspond to problematic code. TypeDevil is complementary to JavaScripts strict mode, which warns about potential programming errors. None of the problems detected by their analysis are found with strict mode

This approach may detect these type inconsistencies by analyzing JavaScript code dynamically. It is applied on different JavaScript files within web applications to show that this approach may find type inconsistencies dynamically. It may find type inconsistency in JavaScript codes that are written in non-strict mode. This approach can only be applied to the JavaScript files and may detect type consistency within the JavaScript files. However, it cannot find inconsistency in AngularJS MVC applications. The reason it that to find both type and identifier inconsistency in MVC application, both the controller JavaScript file and view HTML file should be considered. This approach does not consider the inconsistency between the HTML and JavaScript files.

3.3.2 Aurebesh

There are few popular JavaScript-based MVC frameworks used by the developers such as AngularJS, BackboneJS, EmberJS, and ReactJS. Among all the frameworks AngularJS is the most popular frameworks [30]. The main purpose of using these frameworks is that it abstracts DOM API method calls between the controllers and views. Unfortunately, MVC frameworks are susceptible to inconsistencies between the identifiers and types of variables and functions used throughout the application. Specifically, these frameworks depend on the use of identifiers that represent model objects and controller functions. Defining and using of these identifiers are expected to be consistent throughout all the related models, views, and controllers. The developers have to keep in mind that the values assigned to model objects and returned by controller functions are consistent with their expected types, considering that how it is used. As model objects and controller functions are used to represent major functionalities of the web application, so any inconsistencies between these identifiers and types can potentially lead to a significant loss in functionality and performance. These inconsistencies are difficult to detect when multiple model-view-controller groupings exist in the application. Besides, developers do not get any exceptions and warnings provided in the event if any inconsistency occurs

To detect inconsistency in JavaScript MVC application recently an approach has been proposed by Frolin et al. (2015) [6]. It contains two types of inconsistencies namely identifier consistency and type consistency. This approach may identify both types of inconsistency in JavaScript MVC application. The approach is implemented as a tool named AUREBESH that may detect inconsistency by performing static code analysis. A fault injection study is conducted into 20 open source AngularJS applications considering to be representative of JavaScript MVC application. The researcher found 15 real bugs and 11 error message generated by the AUREBESH. It is observed that among the 15 bugs, 13 bugs are identifier inconsistency and rest of the two bus are type inconsistency. Besides, four patterns are found by analyzing these 15 bugs for example.

- Identifier defined elsewhere (7 cases),
- An Incorrect identifier (5 cases),
- Boolean assigned a string (2 cases)
- Identifier name not updated (1 case).

However, while using custom directives in AngularJS applications, this approach cannot detect inconsistency. The reason is that it omits the presences of custom directives in the AngularJS application. Recently, AngularJS applications are written by following angular style guide [31]. This approach also fails to detect inconsistencies in these applications that were developed according to the angular style guide.

3.4 Survey Study on AngularJS

According to Google trend [26], AngularJS is the most popular JavaScript framework. Besides it has been observed in a study that AngularJS is being grown over the last few years. Until 2015 July, there are 32K Git-hub repositories, 75K Stack Overflow questions, and 120K YouTube videos based and on AngularJS application [14]. In spite of having a lot of resource on AngularJS, there is no clear knowledge of how this framework affects the development experiences of JavaScript software by providing proposed design and features. There are some specific issues that are not clear to the developers such as what are the most important features of AngularJS, what are the common and critical problems faced by developers and which aspects of AngularJS can be improved. In this research work, researchers try to find the answer to these question by conducting a survey study on 460 JavaScript developers [17]. Developers who never used JavaScript MVC frameworks may understand the benefits and common problems associated to use by reviewing the case of AngularJS. Besides developers who have already used AngularJS frameworks, may able to learn how to use this framework more efficiently by following best practices and avoid bad AngularJS programming pastises. The study also reveals the relevant features of the AngularJS such as dependency injection, custom components, two-way data binding, and also the most frequent problem faced by the AngularJS developers, API complexity, etc.

In an early work, it is found that there are two types of inconsistency that may occur in JavaScript MVC framework. The first one is identifier inconsistency, and another is data type inconsistencies. Identifier inconsistencies occur when identifier used in one layer are undefined in the lower layer. Types of values assigned to a variable, or returned by a function that does not match with their use in the view are responsible for data type inconsistencies. The researcher observes that both types of inconsistencies are not easily caught during the development especially working with the multiple layers. The existing tool namely AUREBESH that automatically identify these inconsistencies. However, in this survey study, it is found that only 39% of the respondent consider that silent failure, corresponding to identifier inconsistencies, are real problems during development [17]. Besides 84.5% of the respondent considered that two-way data binding related to type consistency as a valuable feature. These results reflect over the real problem while using AngularJS frameworks.

3.5 Summary

The automatic inconsistency identification technique helps the developers to find the inconsistency among the modules of the applications easily. Inconsistency among the modules creates hidden bugs and developers need to perform a manual inspection to identify the inconsistency. Therefore, research field to automatically identify inconsistency in AngularJS applications is really needed. Various approaches are also developed by the researchers. However, those approaches are not supported to deal with the AngularJS new features and cannot find inconsistency in the presence of custom directives. The proposed technique will be discussed in the next chapter.

Chapter 4

Fantasia: An Automatic Scafolding and Inconsistency Identification Tool for AngularJS 1.x MVC Applicatinos

In this chapter, an approached is proposed that can identify inconsistencies in AngularJS 1.x MVC applications. It is mentioned in the previous chapter that existing approach can detect inconsistencies only for the older version of AngularJS applications. Moreover, it also omits the presence of custom directives in those applications. The recommended angular coding style guides and the new features of AngularJS are not supported by the existing tool. It is officially recommended for the developers to follow recommended coding style guides and new features of AngularJS [2]. It is also recommended to create custom directives while developing loosely coupled modules and applications [2]. So, while identifying the inconsistencies, the presence of the custom directives should be considered. In order to overcome the above limitation of the existing approach, an automatic approach namely FANTASIA is proposed and developed that can identify inconsistencies in AngularJS MVC applications in the presences of custom directives. The proposed approach is exhaustively described in the following section of this chapter. component and AngularJS object it is

4.1 Overview of proposed Inconsistency Identification Techniques

The internal architecture of FANTASIA is shown in the Figure 4.1 . The architecture contains seven small components where each component performs predefine responsibilities. The information flow and the activity of each component are described below.

1. **Route Extractor:**In AngularJS MVC applications, the inconsistencies occur between the correspondent view and controller, primarily it is required to identify the correspondent view and controller files. It is needed to extract the configuration file for getting the controllers and views that are correspondent to each other. The reason is that in the configuration file, the route configurations are defined where the corresponding view and controller names are mentioned. Route Extractor extracts the configuration file and finds the correspondent view and controller module files. Finally, it passed those files into the next component. A procedure namely *GetInitializedMvcGroup* is implemented within the Route Extractor component. The procedure makes an empty mvc group list with the controller, view file names and contents.

2. **Component Extractor:** After getting the required module files, it is the responsibility of the Component Extractor to categorize the files in various modules and read the data of those files. The categorized data is feed by

the next two components namely View Extractor and Controller Extractor.

3. **View Extractor:**View Extractor gets view code from the component extractor. It extracts the view html code and generates the DOM. From the DOM, it finds the model variables and controller functions that are used in the view using an Angular expression or AngularJS built in directives. The accepted data types of AngularJS directives are also gathered where the model variables and controller functions are used. Finally, it finds the custom directives whether that are used in the view. If custom directives are present in the view, it makes a list of custom directives. Then, it provides the list to the Directive Extractor component.

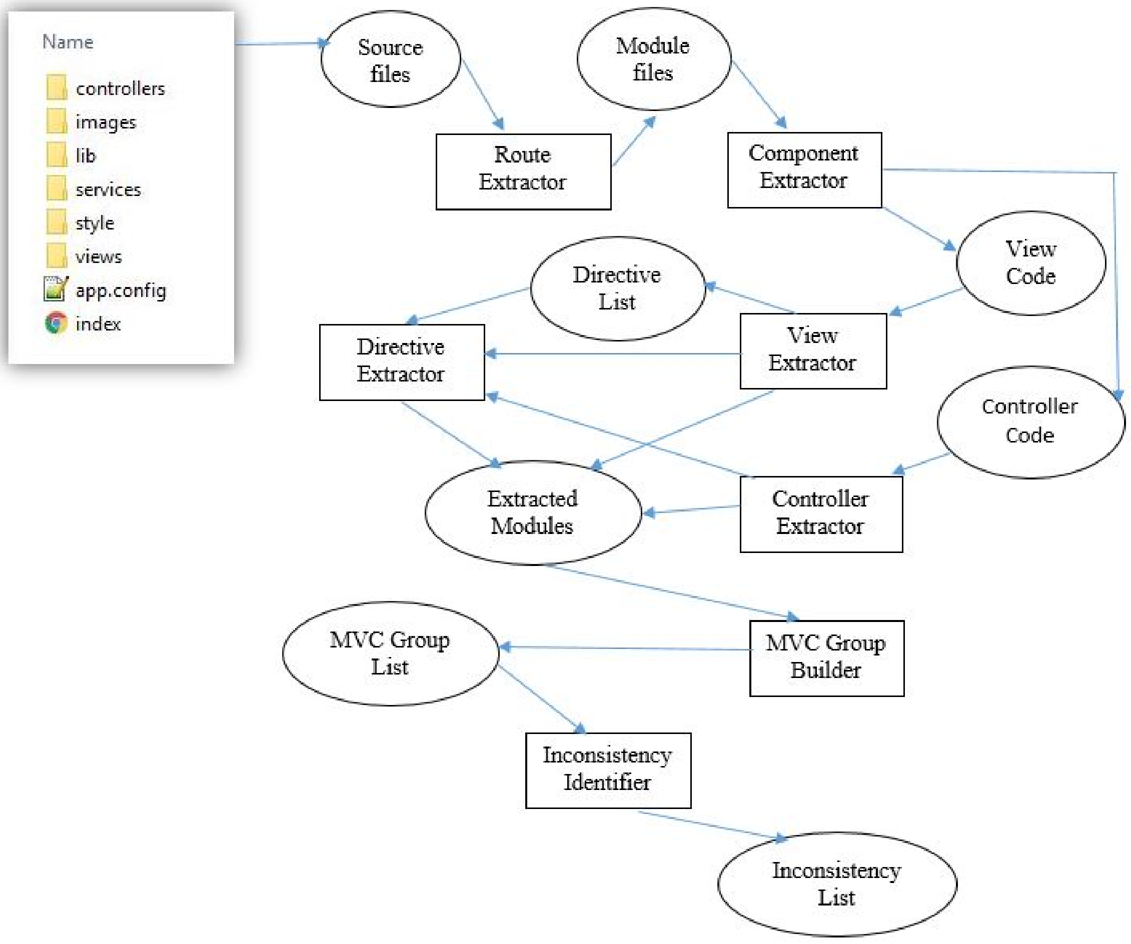


Figure 4.1: High Level Architecture of Fantasia

4. **Controller Extractor:**Controller Extractor also gets the controller code as JavaScript code from the Component Extractor. It generates Abstract Syntax Tree (AST) using the code. After that, it finds all the model variables and the controller functions that are defined in the controller.

5. **Directive Extractor:** Directive Extractor gets the directive list from the View Extractor. It gets the required directive code and finds the type of this directive. Based on the type, it extracts the directive code and gathers the required view and controller that are used in the directive. An algorithm namely Extracting Custom Directive is implemented within the Directive Extractor component. The algorithm extracts the custom directives from its definition file and extracts the associated modules related to this directives.

6. **MVC Group Builder:** MVC group builders takes input from the above described components. It collaborates other components and makes the composite component. It aims to build a list of MVC group where each group contains the correspondent and associated view, model, controller, and directive. It is the complete mvc group that is required in the next step. A procedure namely *GetPopulatedMvcGroup* is implemented by collaborating the View Extractor and Controller Extractor, and Directive Extractor components to build the complete mvc group.

7. **Inconsistency Identifier:** The last component is Inconsistency Identifier. It takes the complete mvc group as input. Finding the inconsistencies in each element of the group is the responsibility of this component. An algorithm namely Identifying Inconsistency is implemented in this component.

It is seen that the proposed technique comprises two procedures and two algorithms. The two procedures namely *GetInitializedMvcGroup* and *GetPopulatedMvcGroup* are used to get the required modules files such as controller, views and directives and to extract those files. The two algorithms namely Extracting

Custom Directives and Identifying Inconsistency are developed to extract the custom directive if it presents in the application. Finally, identify the inconsistency among the correspondent controllers and views. These two procedures and the algorithms are elaborately discussed in the following sections of this chapter.

4.2 Initializing MVC Group

As the proposed technique consider the presence of custom directives; it is required to build the mvc group with related modules appropriately. The technique starts with initializing the mvc group by extracting the routing file or the configuration file

Procedure 1: GetInitializedMVCGroup

```
Input  : A list of files (F) that contains self-descriptive angular modules
Output: A list of MVC Group (groups) where each group contains
        model(m), view(v), controller(c), alice(a) and custom
        directive(dir)
1  begin
2      $groups \longleftarrow \emptyset$;
3      foreach $f \in F$ do
4          if $f.extention = config$ then
5              $Ast \longleftarrow getAST(f)$;
6              $States \longleftarrow getStates(Ast)$;
7              foreach $s \in States$ do
8                  $v.name \longleftarrow findView(s.templateUrl)$;
9                  $c.name \longleftarrow findController(s.controller)$;
10                 $a.name \longleftarrow s.controllerAs$;
11                 $groups \longleftarrow groups \cup group(v.name, c.name, a.name)$;
12             end
13         end
14     end
15 end
```

A list of mvc groups is initialized from the application source files. The list contains several modules like model(m), view(v), controller(c), alice(a) and custom directive(d). The model(m) contains list of model variables with the identifier(id) and data type(type) ,controller(c) contains a list of controller functions with the identifier(id) and return type(type), custom directive (d) contains a list of custom directives used into the view. The view (v) contains the list of all model variables and controller functions identifier (id) and the accepted type (type) of angular expression where these are used.

Procedure 1 starts with initializing empty groups by loop through the source files. It will find the configuration file from the list of source files. An Abstract Syntax Tree (AST) is generated from the configuration file using the getAST() function (Procedure 1,Line:5).The AST contains different states where the related view and controller names are defined. Using the getStates() function (Procedure 1, Line:6) the states are extracted from AST. For every state, view and controller names are found using the findView() and findController() function (Procedure 1, Line:8,9) respectively. A new group is initialized by using the view, controller, alice name and add this into the mvc group list (Procedure 1, Line:11).

4.3 Populating MVC Group

After initializing, it is required to populate the mvc groups with respective model variables and controller functions identifier with their type. Procedure 2 populates the initialized mvc group. It takes the list of files (F) and the initialized mvc group and updates the mvc group with the model variables, controllers, and views. Loop through into the mvc group; it finds the controller files. An Ast is generated for every controller file. The list of model variables, controller functions identifiers and types are extracted from the Ast by using the extractModel() and extractController() (Procedure 2, Line:6,7) functions respectively. The model variables and the controller functions contain those identifiers that are defined by the developers. However, the model variable types and controller function return types are assumed to be a primitive type such as String, Number, Boolean. However, the model variables and the return type of the controller functions may contain complex data type with a complex expression. In this case, the type of model variables and the controller functions are assigned to the unknown type. After that, the procedure continues by finding the view files and generates the DOM using the getDOM() function (Procedure 2, Line:10). The DOM may contain zero or more custom directives. So every custom directive should be identified and added to the groups custom directive list. The view is also extracted from the Dom using the extractView() function (Procedure 2, Line:16) and added to the view list.

Procedure 2: GetPopulatedMVCGroup

```
Input  : A list of files (F) and initialized MVC Group
Output: Updated MVC Group
1  begin
2      foreach $f \in F$ do
3          foreach $g \in group$ do
4              if $f.name == g.c.name$ then
5                  $Ast \longleftarrow getAST(f)$;
6                  $g.m \longleftarrow extractModel(Ast)$;
7                  $g.c \longleftarrow extractController(Ast)$;
8              end
9              if $f.name == g.v.name$ then
10                 $Dom \longleftarrow getDOM(f)$;
11                 if $DomhascustomDirective$ then
12                     foreach $customDirective \in Dom$ do
13                         $g.d.name.add(customDirective)$
14                     end
15                 end
16                 $g.v \longleftarrow extractView(Dom)$;
17             end
18         end
19     end
20 end
```

4.4 Extracting Custom Directives

Each populated mvc group contains, a list of model variables, controller functions, associated view and a list of custom directives. Algorithm 3 updates the mvc group by adding some new group. This new group is created by extracting the custom directives. Algorithm 3 is feed with a list of files and the updated mvc group.

Algorithm 3: Extracting Custom Directives

```
Input  : A list of files (F) and Updated MVC Group
Output: A Complete MVC Group
begin
    $directives \longleftarrow getDirectiveList(F)$;
    foreach $g \in group$ do
        foreach $d.name \in group.d$ do
            $m \longleftarrow \emptyset$, $v \longleftarrow \emptyset$, $c \longleftarrow \emptyset$;
            $file \longleftarrow findDirective(d.name, directives)$;
            $Ast \longleftarrow getAST(file)$;
            $dir \longleftarrow extractDirective(Ast)$
            $Dom \longleftarrow getDOM(dir.templateUrl)$;
            $v \longleftarrow extractView(Dom)$;
            if $dir.scope == false$ then
                $m \longleftarrow group.m$;
                $c \longleftarrow group.c$;
                $a \longleftarrow group.a$;
            end
            else if $dir.scope == true$ then
                $Ast \longleftarrow getAST(dir.controller)$;
                $m \longleftarrow group.m \cup extractMode(Ast)$;
                $c \longleftarrow group.c \cup extractController(Ast)$;
            end
            else if $dir.scope == \{\}$ then
                $Ast \longleftarrow getAST(dir.controller)$;
                $m \longleftarrow extractModel(Ast)$;
                $c \longleftarrow extractController(Ast)$;
                $a \longleftarrow dir.controllerAs$;
            end
            $groups \longleftarrow groups \cup group(m, v, c)$;
        end
    end
end
```

It starts with initializing a directive list using getDirectiveList() function (Algorithm 3, Line:2). It contains all the directive files present in the application. For every mvc group and every directive within the group, a new group is created.The group creation begins with initializing the model(m),view(v)and controller(c) with empty set (Algorithm 3, Line:5). The directive file is found from the directive list (directives) using findDirective() function (Algorithm 3, Line:6). An AST is generated using that file and extract the directive from the AST using the getAST() and extractDirective() functions respectively (Algorithm 3, Line:7,8). The directive contains a property named templateUrl that indicates its view file. The DOM is generated from that file, and the view is extracted from that DOM using the getDOM() and extractView() function respectively (Algorithm 3, Line:9, 10).

The directive contains a property named scope that indicates its controller and model. If the scope is false , it refers that this directive does not manipulate the parent controller and model properties. The parent controller and model properties can directly be used within the directives.So, in this case, the model(m), and controller(c) remain same as its parent model and controller (Algorithm 3, Line:11-14)

If the scope is true, it refers that this directive can prototypically inherit its parent controller and model and can manipulate them. For this case, an AST is generated from the controller file that responsible for this directive (Algorithm 3, Line:17). Model and controller are extracted from the AST using the extractModel() and extractController() respectively (Algorithm 3, Line:18,19). After that, they are merged with their parent model and controller and assigned them to the directives model and controller. When the scope is , it means that the model and controller of this directive are isolated from its parent model and controller. For this directive, new model and controller are created. An AST is generated from its controller file. The new model, controller, and alice are extracted from that AST. After extracting the directive, a new mvc group is created, and after that, it is merged with the updated mvc group. By the end of this algorithm, a complete mvc group is created that is used for next step to identify inconsistency.

4.5 Identifying Inconsistencies

Algorithm 3 provides a complete MVC Group by extracting the custom directives. The approach of Algorithm 4 namely Identify Inconsistency is to compare the model variables and controller functions in the same grouping. It is done to identify the potential inconsistencies that exist within the same grouping. The algorithm starts by searching the inconsistencies related model variables loops through every model variables that are used in the view and controller (Algorithm 4, Line: 5). For such all model variables that are defined in the controller, their identifiers are checked to see if these also exists and are defined to the corresponding

Algorithm 4: Identify Inconsistency

```
Input  : Complete MVC Group
Output: A list of inconsistency (INC)
1  begin
2      $INC \longleftarrow \emptyset$;
3      foreach $g \in group$ do
4          foreach $m \in group.m$ do
5              foreach $v \in group.v$ do
6                  if $m.id \neq v.id$ then
7                      $INC \longleftarrow idMissMatch(m.id, v.id)$;
8                  end
9                  else if $m.type \neq v.type$ then
10                     $INC \longleftarrow typeMissMatch(m.type, v.type)$;
11                 end
12             end
13         end
14         foreach $c \in group.c$ do
15             foreach $v \in group.v$ do
16                 if $c.id \neq v.id$ then
17                     $INC \longleftarrow idMissMatch(c.id, v.id)$;
18                 end
19                 else if $c.type \neq v.type$ then
20                     $INC \longleftarrow typeMissMatch(c.type, v.type)$;
21                 end
22             end
23         end
24     end
25 end
```

view. The checking is done based on string comparison. If it does not exist it means either these variables are not used in the view or their identifiers are inconsistent. So, there exists an identifier inconsistency. So, it is included in the inconsistency list (INC) (Algorithm 4, Line: 7). However, if the model variable exists, next the data type of the model variables are checked into the view and controller. If the data type of the model variables is not the same corresponding to the view and controller, it means that a type inconsistency is present that is also included in the inconsistency list (INC) (Algorithm 4, Line: 9). Following the same process inconsistencies in the controller functions are identified (Algorithm 4, Line: 15-20). It is assumed that model variables with unknown types are matched with all types.

4.6 Summary

The chapter exhibits both the high level and low level architecture of the proposed technique. The internal modules of this technique are briefly described. The information flow and the communication among the modules are also discussed step by step. Two procedures and two algorithms are described exhaustively. The procedures are used to get the required source files and extracted information. Besides, the two algorithms are needed to extract the custom directive and finally identify the inconsistency. It is necessary to apply the methodology of this technique to the experimental data set for measuring the accuracy and the efficiency of the proposed technique Therefore, the next chapter deals with the application of the proposed technique on the various experimental data set. The findings and the result of this experiment are also analyzed in the next chapter.

Chapter 5

Implementation and Result Analysis

The effectiveness of the proposed technique is exhibited in this chapter through applying it on different real life AngularJS MVC applications. Two algorithms and two producers are discussed in the previous chapter. It deals with the proposed technique FANTASIA where the algorithms involved extracting custom directives and identifying inconsistency. Fifteen different AngularJS applications having various size are used as experimental dataset. The effectiveness is measured in terms of number of different types of inconsistencies that can be identified by the proposed technique. The proposed technique is implemented using JavaScript programming language so that it can be run on any platform and browser. The existing technique for inconsistency identification namely AURBESH [6] is also used for comparative analysis with FANTASIA. The experimental demonstration shows how the proposed technique may reduce the limitation of the existing technique. A brief explanation regarding of the implementation environment, experimental dataset and comparative analysis are also provided in this chapter.

5.1 Experimental Setup and Required Tools

This section deals with the appliances that are required for implementing the approach and also for the experimental and comparative analysis. FANTASIA is implemented using JavaScript programing language on top of Node.js framework. Moreover, others tools such as Node.js, Esprima and WebStrom are also needed for implementing FANTASIA which have been addressed as follows.

1. **Node.js:**Node.js is a JavaScript run-time built on Chromes V8 JavaScript engine [32]. However, FANTASIA is platform independent tool since it is developed by JavaScript and Node.js. It is available as a Node.js package [33] that can be install using Node.js Package Manager (npm) on any platform.

2. **Esprima:**Esprima is an open source high performance, standard-compliant JavaScript parser written in JavaScript. It is used to construct Abstract Syntax Tree (AST) from JavaScript source code. It provides AST in different formats such as token based, tree based and syntax based [34]

3. **Escodegen:**Escodegen is a JavaScript code generator. It takes AST that is generated by the Esprima and turn back AST into JavaScript code [35].

4. **Estraverse:**Estraverse is used to traverse and manipulate AST. Traversing AST is difficult as there is no unified interface for getting the children for a given node. Estraverse make it easy by using AST generated by Esprima [36].

5. **WebStrom:**WebStorm is an IDE (Integrated Development Environment) built on top of JetBrains IntelliJ platform and narrowed for web development [37]. The IDE provides support for JavaScript, Node.js, HTML and CSS, as well as their modern successors. It is used to implement the proposed technique based on JavaScript and Node.js Framework [32].

6. **Chrome DevTools:**The Chrome DevTools are a set of web authoring and debugging tools built into Google Chrome [38].

7. **System Configuration:**

   (a) Processor : Intel(R) Core(TM) i3 -3227U CPU @1.90 GHz

   (b) RAM: 4GB

   (c) Operating System : Windows 10 Professional

   (d) System Type : 64-bit Operating System x64-based processor

5.2 Experimental Datasets

In order to measure the accuracy of the FANTASIA, a fault injection study is performed on nine different AngularJS applications such as CafeTownsend,Sudoku,etc [39]. Among those applications, twelve open source applications are selected from a list of MVC applications from AngularJS GitHub page [40] specifically, most of the applications which were chosen to evaluate the existing approach AUREBESH [6]]. All the applications are refactored into the Google recommended coding style guide. Besides, the rest of the three applications are also selected from GitHub repositories that contain custom directives and are also developed by following angular style guides. The size of the selected AngularJS applications is measured by Line of Code (LOC) that refers the combined lines of HTML and JavaScript code, except external libraries.

Table 5.1: List of AngularJS MVC applications (absence of custom directives)

No	Application	Application Category	Size (LOC)
1	Balance Projector	Finance Tracker	511
2	CafeTownsend	Employee Tracker	452
3	Currencies-GH-Page	Currency Converter	350
4	Cyptrograpgy	Encoder	523
5	Event bus	Calculator	690
6	GitHub Contributors	Search	459
7	Kodigon	Encoder	984
8	Life Stacks	Life Style	350
9	Mark Down	Html Compiler	150
10	Memory Game	Puzzle	181
11	Sliding Puzzle	Puzzle	608
12	Sudoku	Game	706

Table 5.2: List of AngularJS MVC applications (having custom directives)

No	Application	Application Category	Size (LOC)
1	Shopping List	Life Style	511
2	Student Management	Management System	452
3	Funny Facebook App	Entertaining	350

5.3 Fault Injection Study

To represent the efficiency of FANTASIA, a fault injection study is performed on the dataset. The injection is performed by initializing a mutation. The mutation is injected to a line of code in the application source code files. In these AngularJS application, the source code files are either the HTML files or JavaScript files. After that, FANTASIA is run on the mutated version of the application and record whether FANTASIA may identify the inconsistency initialized by the mutation. If the inconsistency is identified, the result of the injection is noted as Successful, otherwise Failed. According to Frolin et-al [6], an MVC applications is consistent if it holds the following four properties.

1. The controller and view can only use model variables that are defined in the model.

2. The view only uses controller functions that are defined in the controller.

Table 5.3: Types of Injected Faults

No	Description	Property
1	Change the name of a model variable used in line N of a view	1
2	Change the name of a model variable used in line N of a controller	1
3	For a particular model variable used in line N of a view, remove the declaration of that model variable in a corresponding model	1
4	For a particular model variable used in line N of a controller, remove the declaration of that model variable in a corresponding model	1
5	Change the name of a controller functions used in line N of a view	2
6	For a particular controller function used in line N of a view, remove the declaration of that controller function in a corresponding controller	2
7	For a particular model variable used in the view that expects a certain type T1, change the declaration of that model variable in line N of a corresponding model so that the type is changed to T2	3
8	For a particular model variable used in the view that expects a certain type T1 and declared in line N of a corresponding model, change the expected type to T2 by mutating the ng attribute name	3
9	For a particular controller function used in the view that expects a certain type T1, change the return value of that controller function in line N of the controller to a value of type T2	4
10	For a particular controller function used in the view that expects a certain type T1 and returns a value in line N of a corresponding controller, change the expected type to T2 by mutating the ng attribute name	4

3. The expected types of corresponding model variables in the view match the assigned types in the model or controller.

4. The expected and returned types of corresponding controller functions match in the view and controller

In this experiment, ten types of mutations are considered that are illustrated in Table 5.3. Among these types, every mutation type corresponds to a violation of the above four consistency properties.

The result of detecting mutation type represents how well FANTASIA may detect the violation of corresponding property. For this experiment, at least one injections are performed per mutation type that amounts to 10 to 15 injection per application. It is noted that some mutation types are not applicable for all applications. For example, it is not mandatory that all controllers use the model variables. For these types of specific case, some mutation types are not considered. As a result, in some applications there are less than 15 injections injected. The location of the mutated code is chosen uniformly considering that if the line, where the code is present is applicable for current mutation type. For every injection, the number of successful identifications and number of failed identifications are considered for further result analysis and comparative study.

5.4 Result Analysis

The results of fault injection study are illustrated in Table 5.4 and Table 5.5 respectively. The result of fault injection study on twelve AngularJS MVC applications (without custom directives) is represented by Table 5.4. Besides, Table 5.5 shows the result of fault injection study on three AngularJS MVC applications (having custom directives). For every application the precession and recall is calculated. It shows that FANTASIA performs accurate with the overall recall of 92.05% (Table 5.5). Moreover, FANTASIA also performs precisely with the 85.6% recall in other three applications (Table 5.6) that contain custom directives.

5.4.1 Accuracy

The accuracy of FANTASIA is inferred from both the Table 5.4 and Table 5.5. FANTASIA performs accurately in MVC applications (absent of custom directives) with overall 92.05% recall. It also performed some failure detections. To analyze the reason for getting the failure detection, the result is divided based on the consistency properties. It is found that 6 numbers of failed detections is caused for the type inconsistency in controller functions and model variables. The reason is that it is assumed that the values used in model variables and return by the controller functions are simple expression having primitive types such as Number,

Table 5.4: Fault injection result performed by FANTASIA on twelve AngularJS MVC applications (Without custom directives)

No	Application	Size (LOC)	Total Injection	Successful Detection	Failed Detection	Recall
1	Balance Projector	511	15	13	2	86.67%
2	CafeTownsend	452	15	14	1	093.94%
3	Currencies-GH-Page	350	12	12	0	100%
4	Cyptrograpgy	523	11	10	1	90.91%
5	Event bus	690	12	10	2	83.33%
6	GitHub Contributors	459	14	12	2	85.71%
7	Kodigon	984	15	13	2	86.67%
8	Life Stacks	350	10	10	0	100%
9	Mark Down	150	11	11	0	100%
10	Memory Game	181	10	10	0	100%
11	Sliding Puzzle	608	12	11	1	91.67%
12	Sudoku	706	14	12	2	85.71%
Overall		**5964**	**151**	**138**	**13**	**92.05%**

String, Boolean. FANTASIA cannot detect those inconsistent model variables and controller functions return type that contain complex expression and get data from external database. Moreover, some applications contain filters that is used in the view for customizing the model variable. The presence of the filters is omitted in the implementation of FANTASIA while extracting the components from the view file. This is also another reason for failed detection.

5.4.2 Performance

FANTASIA can also measure the average time to perform the analysis for each application. It takes the average time of 120 milliseconds to find inconsistencies in the experimental dataset. Besides the worst case of 233 milliseconds in the largest application namely Event Bus. From this observation it is inferred that performance in not a considerable issue for FANTASIA.

Table 5.5: Fault injection result performed by FANTASIA on three AngularJS MVC applications (Having custom directives)

No	Application	Size (LOC)	Total Injection	Successful Detection	Failed Detection	Recall
1	Shopping List	356	12	10	2	83.34%
2	Student Management	476	11	9	2	81.82%
3	Funny Facebook App	678	12	11	1	91.64%
Overall		**1510**	**35**	**30**	**5**	**85.6%**

5.5 Comparative Result Analysis

In the previous section, initially the proposed approach FANTASIA is run on the mutated applications to analyze the accuracy and performance. The result of the analysis is shown in Table 5.4 and Table 5.5. Moreover, for comparative result analysis the existing approach AUREBESH is also run on the mutated applications. After that, number of successful and failed identifications is considered for result analysis. The outcome of the existing approach AUREBESH is very poor with overall 0.0% recall in the twelve AngularJS MVC applications. To understand the reason manual inspection are performed. Two reasons are found that responsible for AUREBESH poor performance. The first reason is that these twelve applications are refactored following the recommended angular style guide. Besides, there is a popular feature of AngularJS called controllerAs that is available in the latest version of AngularJS. It provides an alice for the controller to represent its model variables and controller functions in the view. Since AUREBESH is developed by omitting this feature, it cannot detect the mutated inconsistencies. On the other hand, accuracy of the AURBESH on three AngularJS applications (having custom directives) is also poor. In this case the reason is obvious because AURBESH does not consider the presence of custom directives. So, AURBESH fails to detect inconsistency that present in the custom directives. Since the refactored application are not compatible dataset for AUREBESH, the comparison between the AUREBESH and FANTASIA cannot be performed. To conduct the

Table 5.6: Comparative result between FANTASIA and AUREBESH on twelve AngularJS MVC applications (Without custom directives) :SD represents Successful detection, FD represents Failed detection

No	Application	Total Injection	FANTASIA			AUREBESH		
			SD	FD	Recall	SD	FD	Recall
1	Balance Projector	15	13	2	86.67%	12	3	80.00%
2	CafeTownsend	15	14	1	093.94%	14	1	93.94%
3	Currencies-GH-Page	12	12	0	100%	12	0	100%
4	Cyptrograpgy	11	10	1	90.91%	10	1	90.91%
5	Event bus	12	10	2	83.33%	10	2	83.33%
6	GitHub Contributors	14	12	2	85.71%	13	1	92.85%
7	Kodigon	15	13	2	86.67%	13	2	86.67%
8	Life Stacks	10	10	0	100%	10	0	100%
9	Mark Down	11	11	0	100%	11	0	100%
10	Memory Game	10	10	0	100%	10	0	100%
11	Sliding Puzzle	12	11	1	91.67%	11	1	91.67%
12	Sudoku	14	12	2	85.71%	11	3	78.57%
Overall		**151**	**138**	**13**	**92.05%**	**137**	**14**	**91.49%**

comparative study, the applications are turned into its actual version using the older version of AngularJS omitting the recommended angular style guides. It means that the new features are not present within the applications. Now, FANTASIA and AUREBESH both tools are run on those applications. The number of successful and failed identifications are counted to calculate recall. Table 5.6 shows the comparative result of the study on twelve applications. It is seen that FANTASIA and AUREBESH perform almost same with 92.05% and 91.49% recall respectively. The reason is that FANTASIA is compatible for both the older and new version of AngularJS. It consider both the presence and absence of the new features of AngularJS.

Table 5.7 shows the comparative result of the study between AUREBESH and FANTASIA on three applications having custom directives. It is seen from the table that FANTASIA performs same as it performed before with 85.6% recall. However, AUREBESH performs poor with 42.68% recall since it omits the presence of custom directives no matter which version of AngularJS is used in the

Table 5.7: Comparative result between FANTASIA and AUREBESH on three AngularJS MVC applications (Having custom directives) :SD represents Successful detection, FD represents Failed detection

No	Application	Total Injection	FANTASIA			AUREBESH		
			SD	FD	Recall	SD	FD	Recall
1	Shopping List	12	10	2	83.34%	5	7	41.67%
2	Student Management	11	9	2	81.82%	5	6	36.37%
3	Funny Facebook App	12	11	1	91.64%	6	6	50%
Overall		**35**	**30**	**5**	**85.6%**	**15**	**20**	**42.68%**

applications. It can identify some inconsistencies in these three applications. The reason is that these inconsistencies are not present within the custom directives. AUREBESH can only identify those inconsistencies that only present in the controller and view modules of the applications.

Chapter 6

Discussion and Conclusion

6.1 Discussion and Conclusion

The presence of the inconsistency in AngularJS MVC applications produce hidden bugs, reduce the readability and maintainability of the code. In this research work, an automatic inconsistency identification approach and tool named FANTASIA is proposed. The proposed approach can identify inconsistency in AngularJS MVC application in the presence of custom directives. It has been observed that FANTASIA performs with an overall recall of 92.05% in detecting inconsistency in those AngularJS applications that are developed using recommended style guide and do not contain custom directives. Besides, FANTASIA also performs efficiently to detect inconsistency with an overall recall of 85.6% in that AngularJS MVC applications that contain custom directives. This chapter concludes the research work with a brief description of the threats along with the future direction of this proposed techniques.

6.2 FANTASIA: The proposed inconsistency identification technique in the presence of custom directives in AngularJS MVC Application

FANTASIA comprises two procedures and two algorithms. The two procedures namely, *GetInitializedMVCGroup* and *GetPopulatedMVCGroup* help to initialize the empty mvc groups and populate the mvc groups from the application structure. Moreover, the two algorithms namely Extracting Custom Directives and Identifying Inconsistencies are used to extract the directives and detect to inconsistency in the mvc groups. GetInitializedMVCGroups procedure finds all the required files and initializes the empty mvc groups. After that, procedure GetPopulatedMVCGroups extracts models, views and controllers from those required files. It is also done by extracting the application configuration file where the routing configuration is defined .The first proposed algorithm Extracting Custom Directives finds all the custom directives present in the application and extract them according to their properties. It may updates the previous mvc groups or it may add new mvc group based on its scope properties. Finally, the Identifying Inconsistencies algorithm checks the consistency among the models, views, controllers and custom directives that are present in the mvc groups. In brief, by incorporating these two procedures and two algorithms FANTASIA first initializes an empty mvc group. This group is populated by extracting the controllers and views files. Later, if any custom directive present in the application, they are extracted and updated the mvc groups. Finally, inconsistency is identified within that mvc groups.

6.3 Discussion of the Result

A fault injection study and comparative study are performed on several AngularJS MVC applications to check the accuracy and efficiency of FANTASIA. For the experiment, FANTASIA and existing approach AUREBESH both are implemented using JavaScript programing language. Faults are injected into the selected application according to the JavaScript MVC framework consistency models properties [6]. Both the techniques are run on the same faulty applications, and manual inspection is also performed on those applications. It is observed that FANTASIA performs well with the 92.05% recall in those applications that do not contain custom directives along with developed by following angular style guides. It can also identify inconsistency the rest of the three applications with the recall of 85.6% where the presence of custom directives is considered. It is seen that AUREBESH cannot identify inconsistency in that applications that are developed by following AngularJS latest version. For conducting a comparative study, both FANTASIA and AUREBESH are run on applications that are developed by following the older version of AngularJS .The result shows that both the FANTASIA and AUREBESH perform well with the recall of 92.05% and 91.49% respectively. It is noted that recall of identifying inconsistency not changed for FANTASIA as it can identify inconsistency in both versions of AngularJS. Besides, FANTASIA can detect inconsistency in the presence of custom directives and gets 85.6% recall. On the other hand , AUREBESH cannot identify inconsistency that are present within the custom directives and gets 42.68%.

6.4 Threats to Validity

Although FANTASIA performs better than the existing tool AUREBESH for the particular experiment in this research work, it contains the notable threat that should be considerable. In this section, the threats that may affect the validity of the proposed techniques are discussed.

- **Internal Threat:** The implementation of the techniques and the experimental setup of the proposed techniques are based on JavaScript programing languages as well as HTML. This may cause internal threats that affect the validity of the experimental result. Therefore, the expected result gained through analyzing the experimental projects may vary if the implementation and the experimental projects depend on other platform and technologies.

- **External Threat:** The experimental applications that are chosen are used in the existing techniques may affect the experiment result. Besides, it is assumed that those projects are also elected which are developed following recommended coding guidelines as well as having custom directives. So, if the selected applications change the assumptions, it may affect the experimental results.

- **Construct Threats:** To analyze the effectiveness of the proposed techniques, Precision and Recall software metrics is used in this experiment. The experimental results are analyzed based on the number of faults detected by the proposed and existing tool and represent their efficiency using precision recall. Therefore, analyzing the result in different metrics effect the generalization of the expected result.

6.5 Future Work

The idea of identifying inconsistency in AngularJS MVC application in the presence of custom directives contributes to the literature by inventing techniques named FANTASIA. The techniques can be incorporated with the existing tool AUREBESH to identify inconsistency to other JavaScript MVC frameworks. In this research work, the scope of this proposed technique is just to identify the inconsistency. However, it can be further used to automatically remove the inconsistency from AngularJS MVC applications as an inconsistency detection tool. The proposed technique FANTASIA is implemented using JavaScript programing language and Node.js framework, in the form of Command Line Interface (CLI). It is completely a platform independent tool since Node.js can be run on all platform . However, the technique can be implemented using other programing languages such as C++, Java, C#, etc. Moreover, a Google Chrome extension is being built to detecting inconsistency in AngularJS MVC applications based on the proposed techniques. The AngularJS applications that are used for fault injection study and result analysis are not too larger considering the Line of Code (LOC) and a number of modules. Therefore, dealing with the real life applications or industrial applications is a future issue.

Bibliography

[1] M. S. Mikowski and J. C. Powell, "Single page web applications," *B and W*, 2013.

[2] AngularJS, "AngularJS Site." `https://docs.angularjs.org/`, 2016. [Online; accessed 8-August-2016].

[3] V. Balasubramanee, C. Wimalasena, R. Singh, and M. Pierce, "Twitter bootstrap and angularjs: Frontend frameworks to expedite science gateway development," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–1, IEEE, 2013.

[4] F. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah, "An empirical study of client-side javascript bugs," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 55–64, IEEE, 2013.

[5] M. Pradel, P. Schuh, and K. Sen, "Typedevil: Dynamic type inconsistency analysis for javascript," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 314–324, IEEE, 2015.

[6] F. S. Ocariza Jr, K. Pattabiraman, and A. Mesbah, "Detecting inconsistencies in javascript mvc applications," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pp. 325–335, IEEE Press, 2015.

[7] S. Seshadri and B. Green, *AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps.* " O'Reilly Media, Inc.", 2014.

[8] trygver, "trygver." `http://heim.ifi.uio.no/~trygver/index.html`, 2016. [Online; accessed 12-September-2016].

[9] G. E. Krasner, S. T. Pope, *et al.*, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.

[10] codinghorror, "understanding-model-view-controller." `https://blog.codinghorror.com/understanding-model-view-controller/`, 2016. [Online; accessed 12-September-2016].

[11] DocForge.com, "Framework." `http://docforge.com/wiki/Framework`, 2016. [Online; accessed 12-September-2016].

[12] DocForge.com, "Web application framework." `http://docforge.com/wiki/Web_application_framework`, 2016. [Online; accessed 12-September-2016].

[13] jquery.com, "jquery.com." `https://jquery.com/`, 2016. [Online; accessed 12-September-2016].

[14] docforge, "Model-View-Controller." `http://web.archive.org/web/20151025035024/http://docforge.com/wiki/Model-View-Controller`, 2016. [Online; accessed 12-August-2016].

[15] stackoverflow.com, "what-is-separation-of-concerns." `http://stackoverflow.com/questions/98734/what-is-separation-of-concerns`, 2016. [Online; accessed 12-September-2016].

[16] K. Hrgovic, "Top 10 Most Used JavaScript Frameworks." `https://blog.codeanywhere.com/top-10-most-used-javascript-frameworks/`, 2016. [Online; accessed 12-September-2016].

[17] M. Ramos, M. T. Valente, R. Terra, and G. Santos, "Angularjs in the wild: A survey with 460 developers," *arXiv preprint arXiv:1608.02012*, 2016.

[18] sitepoint, "practical-guide-angularjs-directives-part-two." `https://www.sitepoint.com/practical-guide-angularjs-directives-part-two/`, 2016. [Online; accessed 12-September-2016].

[19] angularjs, "Developer Guide." `https://docs.angularjs.org/guide`, 2016. [Online; accessed 12-September-2016].

[20] sitepoint, "introduction-angularjs-style-guides." `https://www.sitepoint.com/introduction-angularjs-style-guides/`, 2016. [Online; accessed 12-September-2016].

[21] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pp. 118–127, IEEE, 2001.

[22] B. Taraghi and M. Ebner, "A simple mvc framework for widget development," in *Proceedings of the International Workshop on Mashup Personal Learning Environments (MUPPLE). CEUR-WS*, pp. 38–45, 2010.

[23] J. Fujima, "Building a meme media platform with a javascript mvc framework and html5," in *Webble Technology*, pp. 79–89, Springer, 2013.

[24] J. Coutaz, "Pac, an object oriented model for dialog design," in *Proceedings Interact*, vol. 87, pp. 431–436, 1987.

[25] Y. Tanaka, *Meme media and meme market architectures: Knowledge media for editing, distributing, and managing intellectual resources.* John Wiley & Sons, 2003.

[26] G. Trends, "JavaScript Framwork." `https://www.google.com/trends/`, 2016. [Online; accessed 12-August-2016].

[27] F. S. Ocariza Jr, K. Pattabiraman, and B. Zorn, "Javascript errors in the wild: An empirical study," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pp. 100–109, IEEE, 2011.

[28] F. S. Ocariza Jr, K. Pattabiraman, and A. Mesbah, "Autoflox: An automatic fault localizer for client-side javascript," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 31–40, IEEE, 2012.

[29] F. S. Ocariza Jr, K. Pattabiraman, and A. Mesbah, "Vejovis: suggesting fixes for javascript faults," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 837–847, ACM, 2014.

[30] ImProgrammer, "Popular JavaScript Framwork." `http://www.improgrammer.net/most-popular-javascript-frameworks-2015/`, 2016. [Online; accessed 12-August-2016].

[31] GitHub, "GitHub/AngularJS Style Guide." `https://github.com/johnpapa/angular-styleguide`, 2016. [Online; accessed 9-August-2016].

[32] NodeJs, "Node JS Site." `https://nodejs.org/en/`, 2016. [Online; accessed 8-August-2016].

[33] NPM, "NPM Site." `https://www.npmjs.com/`, 2016. [Online; accessed 8-August-2016].

[34] Esprima, "Esprima Site." `http://esprima.org/`, 2016. [Online; accessed 8-August-2016].

[35] Escodegen, "Escodegen Site." `https://github.com/estools/escodegen`, 2016. [Online; accessed 8-August-2016].

[36] Estraverse, "Estraverse Site." `https://github.com/estools/estraverse`, 2016. [Online; accessed 8-August-2016].

[37] Webstorm, "Webstorm Site." `https://www.jetbrains.com/webstorm/`, 2016. [Online; accessed 8-August-2016].

[38] Chrome, "Chrome DevTools." `https://developer.chrome.com/devtools`, 2016. [Online; accessed 8-August-2016].

[39] builtwith, "builtwith.angularjs.org." `https://builtwith.angularjs.org/`, 2016. [Online; accessed 8-August-2016].

[40] GitHub, "GitHub/AngularJS." `https://github.com/angular/angular.js`, 2016. [Online; accessed 9-August-2016].