AngularJS in the Wild: A Survey with 460 Developers

Miguel Ramos
Marco Tulio Valente
UFMG, Brazil
{miguel,mtov}@dcc.ufmg.br

Ricardo Terra
UFLA, Brazil
terra@dcc.ufla.br

Gustavo Santos
RMoD Team, INRIA, France
gustavo.santos@inria.fr

Abstract

To implement modern web applications, a new family of JavaScript frameworks has emerged, using the MVC pattern. Among these frameworks, the most popular one is ANGULARJS, which is supported by Google. In spite of its popularity, there is not a clear knowledge on how ANGULARJS design and features affect the development experience of Web applications. Therefore, this paper reports the results of a survey about ANGULARJS, including answers from 460 developers. Our contributions include the identification of the most appreciated features of ANGULARJS (e.g., custom interface components, dependency injection, and two-way data binding) and the most problematic aspects of the framework (e.g., performance and implementation of directives).

Categories and Subject Descriptors D.3.3 [*Frameworks*]

Keywords JavaScript, AngularJS, MVC frameworks.

1. Introduction

JavaScript is a fundamental piece of modern Web applications. It was initially designed as a scripting language to extend web pages with small executable code. However, the language is used nowadays to construct a variety of complex systems (Kienle 2010; Silva et al. 2015). As a result, we are observing the birth of new technologies and tools—including JavaScript libraries and frameworks—to solve common problems faced in the development of such applications. For example, frameworks following the Model-View-Controller (MVC) architecture pattern (or variations of it) are widely used nowadays, including systems such as ANGULARJS, BACKBONE.JS, and EMBER.JS. Among these frameworks, ANGULARJS is probably the most popular one. This fact is evidenced by comparing the number of Google searches (the most queried framework since 2013), the number of contributors in GitHub and the increasing number of questions and answers in Stack Overflow (the framework with more Q&A since mid-2013).

However, despite the increasing practical interest on ANGULARJS, there is no clear knowledge on how the design and features proposed by this framework affect the development experience of JavaScript software. More specifically, it is not clear what are the most appreciated features of ANGULARJS, what are the main problems faced by developers when using the framework, and which aspects of ANGULARJS can be improved. Answers to these questions are important to different developers. First, developers *who use* ANGULARJS can learn how to improve this usage and also how to avoid bad ANGULARJS programming practices. Second, developers *who do not* use JavaScript MVC frameworks can understand the benefits and problems related to these frameworks, by reviewing the case of ANGULARJS. Third, MVC framework builders can use our results to design more powerful and usable frameworks.

This paper reports the results of a survey with 460 developers, when we collected their perceptions about ANGULARJS. We reveal the relevant features of the framework, e.g., custom components, dependency injection, and two-way data binding. We also shed light on the most frequent problems faced by ANGULARJS developers, e.g., due to the complexity of the API to declare directives.

The remainder of the paper is organized as follows. Section 2 introduces ANGULARJS. Section 3 documents the survey design and Section 4 presents the survey results. Threats to validity are presented in Section 5. Section 6 discusses related work and Section 7 concludes.

2. AngularJS in a Nutshell

In this section, we briefly describe the key components of ANGULARJS. A basic understanding of these components is important to interpret our survey results.

Modules: An ANGULARJS application is a set of modules, which act as containers to the different parts of the application. Modules can also depend on other modules.

Services: ANGULARJS services are objects that encapsulate code related with a specific concern. They are instantiated

PLATEAU'16, November 1, 2016, Amsterdam, Netherlands

http://dx.doi.org/10.1145/3001878.3001881

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

Listing 1. Template sample

only once by factories or constructor functions. The created singleton object is shared by the components that depend on it (e.g., controllers, directives, filters, and other services). Typically, ANGULARJS services are stateless objects with a set of methods that deal with specific concerns, such as server requests, manipulation of arrays, asynchronous operations, etc. ANGULARJS also provides built-in services to deal with common concerns in Web applications, such as `$http`, `$filter`, and `$timeout`.

Templates: In Web applications, HTML documents are parsed to generate the DOM (Document Object Model), which is the data structure that models the final document presented to users. ANGULARJS supports DOM-based templates, which are written in HTML and contain proprietary elements and attributes to render the dynamic interface of web applications. Listing 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).

Directives: Directives are specific HTML markers used in templates to define the UI behavior. In Listing 1, the `ng-app` attribute (line 1) is an ANGULARJS directive that specifies the root node of the application. When directives are executed by ANGULARJS they can modify the DOM structure and also register event handlers. During the compilation process, ANGULARJS traverses the DOM and searches for all directives. The directives are executed in order of priority generating the final DOM presented to users. 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 a completely new scope. It is also possible to create custom directives.

Expressions: ANGULARJS expressions are delimited by double curly brackets (`{{expression}}`) or are the values of some directive attributes. Literals (e.g., arrays (`[]`), objects (`{}`)), operators, and variables are examples of elements that 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 `$rootScope`. Since directives can define different scopes from `$rootScope`, expressions from different parts of the template may be evaluated under different scopes. When the value of an expression changes, ANGULARJS updates the view accordingly.

Controllers: ANGULARJS controllers are used to initialize the state of an application and provide an interface to update it. Controllers are used with the `ngController` directive. When this directive is used in a template, it receives the name of a controller and the scope created by the directive is passed as a parameter to the specified controller. The controller must populate the scope object with properties and methods that are used when evaluating expressions.

Digest Cycle: ANGULARJS constantly 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*.

3. Survey Design

First, we performed a *Mapping Study* (Section 3.1) to gather information about ANGULARJS. We relied on the results of this first study to construct the survey. The survey participants were selected among Stack Overflow users (Section 3.2).

3.1 Mapping Study

We use a mapping study to gather information about the use of ANGULARJS. A mapping study is more flexible than a systematic review and it is recommended for studying emergent fields or technologies (Wohlin et al. 2012). For example, information about ANGULARJS is primarily found in blogs, forums, and Q&A sites. Therefore, using the Google search engine, we initially focused on finding documents reporting the benefits and disadvantages of ANGULARJS. To this purpose, we used search queries such as "*the best features of AngularJS*" or "*the bad parts of AngularJS*". We also used two other strategies to reach more sources of information. First, we recursively accessed the references in the sites already reviewed (a practice called "snowballing"). Second, we performed additional search queries for frequently mentioned topics. As an example, we searched for "*transclusion directives*", due to the frequency of references to this topic as a complex ANGULARJS concept. During the revision of blogs, we only considered well-written posts, by authors with experience in software development. A complete list of the posts we consider is available in a companion website.[1] Finally, these documents were analyzed and classified by identifying topic trends, which were used for the survey design.

[1] *https://github.com/aserg-ufmg/angularjs-survey*

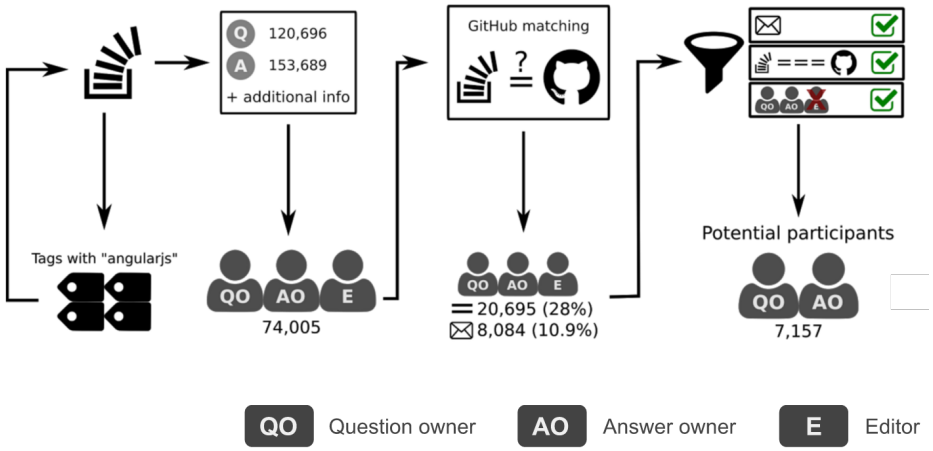


Figure 1. Selecting the survey participants

3.2 Survey Construction and Participants

As a result of the mapping study, we built a 15-minute survey with 25 questions, organized in seven sections: (1) background of the participants, (2) key characteristics of ANGULARJS, (3) problems with ANGULARJS templates, (4) debugging and testing, (5) development practices, (6) complex concepts and features, and (7) ANGULARJS 2.0. To avoid unreliable responses, we asked developers to skip the questions they did not feel confident to answer. All the scales used in the survey have an even number of points, to force participants to make a choice. In multiple choice questions, when the respondents could provide a response not included in the set of answers, we included an *Other* option.

To find participants, we used the Stack Exchange API[2] to search for ANGULARJS developers in the Stack Overflow community. As illustrated in Figure 1, we retrieved all existing tags with the substring "angularjs". We retrieved 120,696 questions and 153,689 answers containing these tags. Next, we extracted data about three types of users: users who own a question (QO); users who own an answer (AO); or users who edited a question or answer (E). In this way, we collected a total of 74,005 users. To find their e-mail address, we matched each user nickname at StackOverflow with an equivalent nickname at GitHub. In this way, we obtained a total of 20,695 matched users (28%). However, only 8,084 users (10.9%) had public contact information at GitHub. In a last step, 7,157 users were marked as potential participants in the survey, because they have valid e-mail address and at least one operation with the collected answers that is not a simple edition. These users were ranked considering the number of ANGULARJS-related questions and answers at Stack Overflow. Each user received the following score: $S = QO + 3 * AO$, where QO is number of questions owned by the user and AO is the number of answers he owns. We gave an additional weight to answers since users who provide answers tend to have more experience than those who are looking for them.

We randomly selected 30 users from the middle of the rank, with scores between 9 and 24, to run a pilot survey. Their feedback helped us to correct typographical errors and ambiguities in some questions. However, the most important change was in the structure of some questions. In the initial survey version, we used ranking questions, when survey respondents have to rank a list of items in order of importance. After this pilot study, we decided to change to rating questions (when respondents just have to rank each item in a scale ranging from 1 to 4) because some participants complained about the time to answer ranking questions. Due to these changes in the survey, the responses obtained during the pilot phase were discarded.

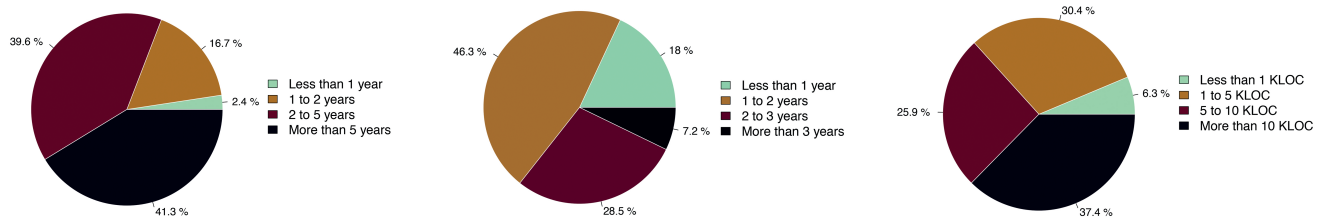
The final version of the survey—which is also available in our companion website—was first sent to a group of 60 users. This time, we received complete responses and an improved response rate. Therefore, we extended the invitation to the rest of the users by daily emailing groups of nearly 700 users from the top of the ranking. At one point, we decided to stop due to replies from developers saying that they had a limited experience with ANGULARJS. In total, we sent the survey to 3,060 users with score between 3 and 831. We received 460 complete responses, representing a response rate of 15%. The survey was open during approximately one month (from early November to early December 2015).

4. Survey Results

4.1 Background

Figure 2 reveals the participants background. The majority of the respondents (97.6%) have at least one year of experience in JavaScript (Figure 2(a)), and 74.8% have one to three years of experience with ANGULARJS (Figure 2(b)). For 37.4% of the developers, the largest application implemented with ANGULARJS has more than 10 KLOC (Fig-

[2] *https://api.stackexchange.com*



(a) JavaScript experience (b) AngularJS experience (c) Largest application

Figure 2. Respondents' background

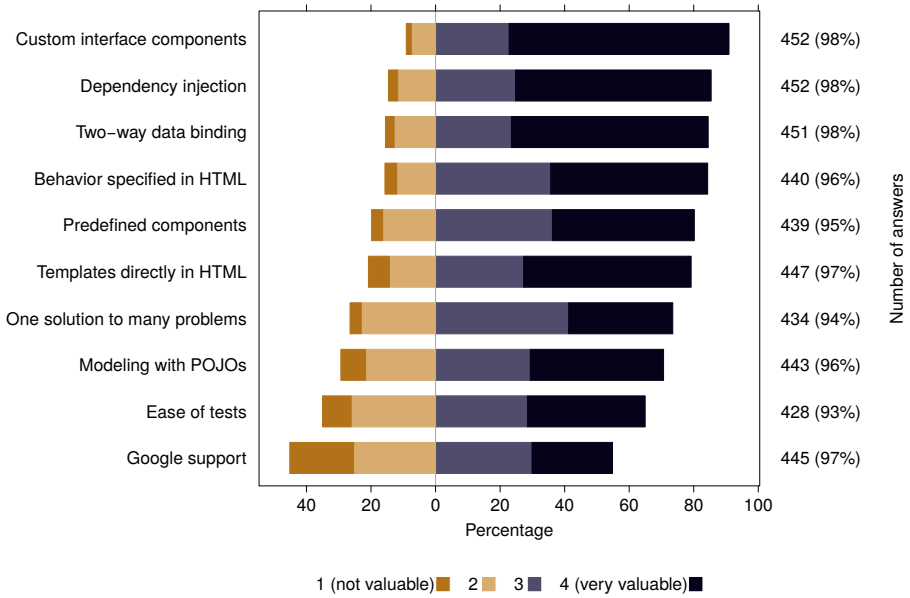


Figure 3. Key features and characteristics of AngularJS

ure 2(c)). Therefore, we can conclude that at least the participants are not novice ANGULARJS developers.

4.2 Key Characteristics and Features of AngularJS

We asked developers about the the following features:

1. *Pre-defined components for code organization:* ANGULARJS has different components to modularize code, which may help in separation of concerns.
2. *Dependency injection:* This design pattern is used by ANGULARJS to manage dependencies between components, to reduce coupling and increase testability.
3. *Use of POJOs in model components:* In ANGULARJS, models are implemented using Plain Old JavaScript Objects (POJOs). There is no need to extend proprietary classes, for example, to provide accessor methods.
4. *Templates in HTML:* ANGULARJS uses DOM-based templates to simplify data binding operations, event mapping, and updating of large interface components.
5. *Support to custom components:* Custom directives can be used as a DSL to define reusable UI components.
6. *Ease of writing tests:* ANGULARJS provides the `ngMock` module to simulate logging operations, HTTP reqs, etc.
7. *Two-way data binding:* ANGULARJS provides synchronization between data in the view and in the model.
8. *Use of HTML to declare UI behavior:* The UI, including its behavior, is defined in standard HTML documents.
9. *One solution to manage many problems:* ANGULARJS is an "opinionated framework", meaning that its design handles common decisions related with Web apps.
10. *Supported by Google:* This support may guarantee the evolution and constant maintenance of the project.

Figure 3 reports the value given by developers to these characteristics and features, ranging from *not valuable* (score 1) to *very valuable* (score 4). At the right of the charts, we include the number of answers of each item. Each individual item was rated by at least 93% of the participants. The top-3 features with more positive scores are the support to *custom components*, the use of *dependency injection*, and the support for *two-way data binding*. The characteristic

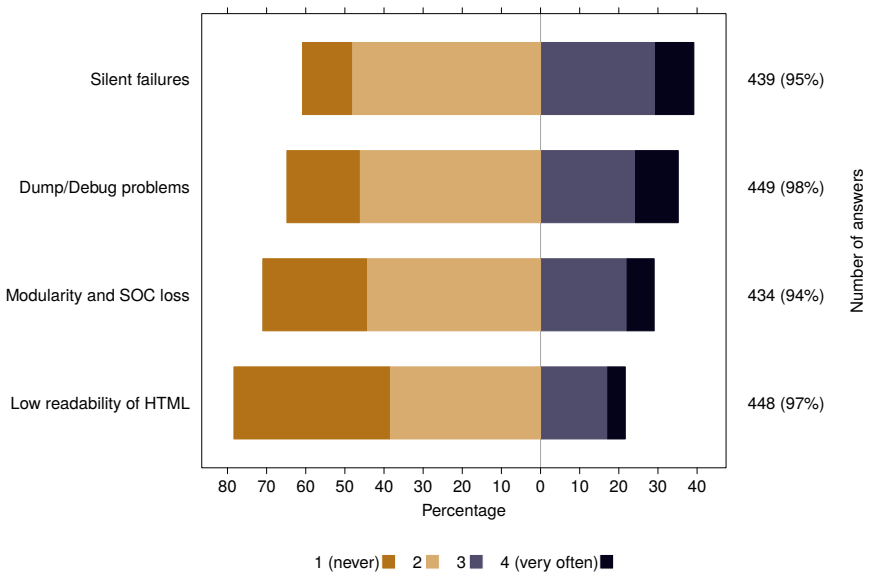


Figure 4. How often these items represent a real problem caused by using code in ANGULARJS templates?

with the lowest number of positive answers is *Google support*, with 45.2% of the respondents seeing it as having no value (score 1) or limited value (score 2).

4.3 Code in HTML Templates

In the mapping study, we detected four possible problems related to placing code in ANGULARJS templates:

1. *Silent failures:* In ANGULARJS, when undefined functions or objects are used in templates no exceptions are raised. As a consequence, applications might fail silently.
2. *Code hard to debug:* Since the code used in ANGULARJS templates is not pure JavaScript, it is not possible, for example, to define break points.
3. *Low readability:* ANGULARJS code might be spread all over the HTML document, hindering readability.
4. *Modularity and Separation of Concerns:* The use of large amount of JavaScript code in HTML is often seen as bad smell (Nguyen et al. 2012), which might hinder separation of concerns.

We asked developers whether these issues are real problems in their daily development. In this case, the score 1 means the issue was never a problem and a score 4 means it is a very frequent problem. As shown in Figure 4, none of the issues have a major detrimental impact, according to the respondents; they have at least 60% of the answers in the low part of the scale (scores 1 or 2).

In a separate question, we asked if the developers had at least once used large amounts of logic in HTML templates and 26.3% of them answered positively. Since this is not a recommended practice, we asked them to indicate possible reasons for their decision The two most voted reasons are ANGULARJS design (54.8%) and the lack of experience in ANGULARJS (43.5%). Lack of experience in Web architecture and in JavaScript were also voted (20.9% and 4.3%, respectively). We also gave the respondents the possibility to indicate other reasons, which were provided by 31 developers. These reasons include, for example, tight deadlines, special cases, easiness or laziness, prototyping purposes, etc.

4.4 Testing

First, we asked the participants to rate the frequency they make use of mocking (provided by the `ngMock` module) when testing their systems. From 441 answers, 72.8% indicated they never or rarely use this module (scores 1 or 2). Possible reasons for this result include limited usefulness of `ngMock` features, unfamiliarity with the module, and few developers putting testing into practice. We also asked the participants to rank how complex is testing ANGULARJS components, from very easy to very difficult. As presented in Figure 5, *services*, *filters*, *controllers*, and *providers* received the higher number of answers with scores 1 and 2. The reason is that these components are very common and usually do not include complex code or code that deals with complex APIs. For example, most code in filters only make string transformations. By contrast, the two components more difficult to test are *transclusion directives* and *directives with external templates*. Probably, developers find these directives more difficult to test because they demand a deeper understanding of ANGULARJS concepts. For example, when creating directives, there are different types of transclusion, different types of scopes, and different ways to interact with the DOM API.

4.5 Complex Concepts and Features

We asked the participants to evaluate several characteristics of ANGULARJS, which were originally identified in the

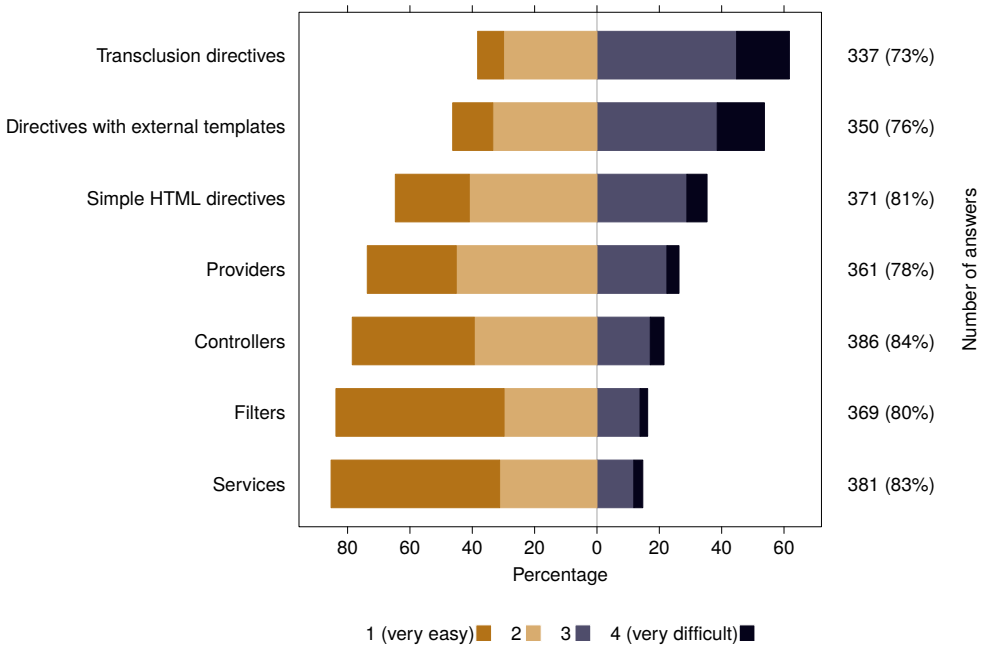


Figure 5. How difficult is it to test these ANGULARJS components?

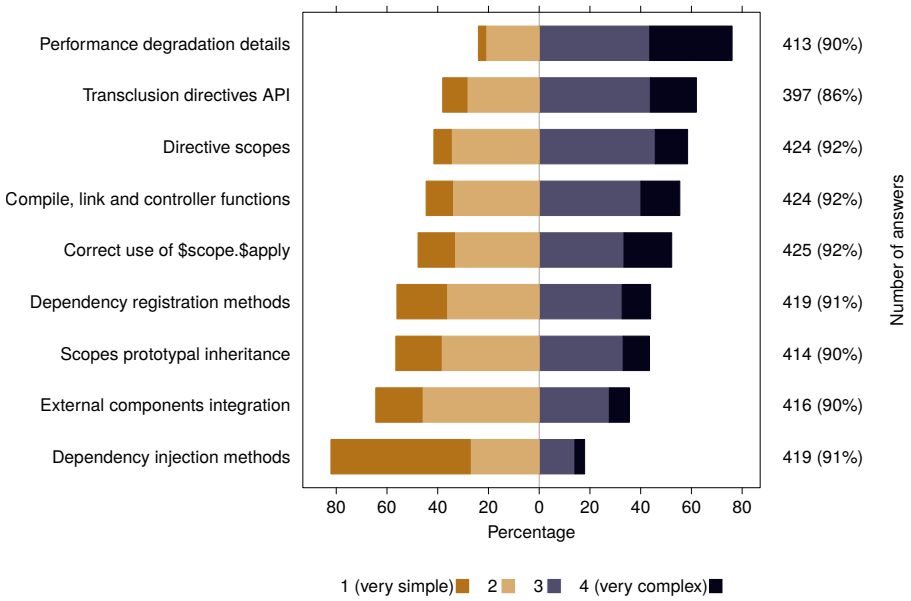


Figure 6. How complex are the the following ANGULARJS aspects and features?

mapping study as complex concepts. The following characteristics were proposed: (1) use of different scopes in directives; (2) use of prototypes to simulate scope inheritance; (3) different types of entities to register dependencies; (4) compile, link, and controller functions (necessary to implement DOM-related directives); (5) transclusion directives; (6) correct use of `$scope.$apply` to manually trigger the digest cycle; (7) tackling of performance degradation details; (8) integration with external components and plug-ins; and (9) correct usage of the syntax to inject dependencies (in order to avoid unexpected results when minifying the code).

Figure 6 summarizes the developers' classification from very simple concept (score 1) to very complex one (score 4). Tackling all the details that can lead to performance degradation was rated by 76% of the participants as a complex task. The next three items in terms of complexity are transclusion directives, the different scopes that can be used when building directives, and the correct use of the functions `compile`, `link`, and `controller`. All these items are somehow re-

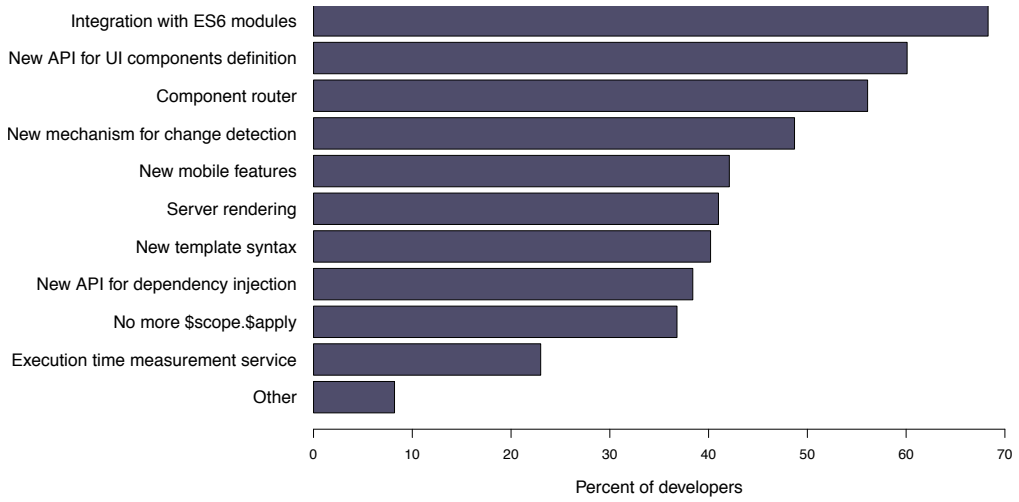


Figure 7. Most expected ANGULARJS 2.0 features

lated with the implementation of directives. The remaining items were rated as more simple than complex, mainly the integration with external components and the use of the correct syntax to inject dependencies.

In a separate question, we asked the developers about the frequency they create directives. As a result, 83.5% of the participants answered they often or very often create their own directives. Despite this fact, many tasks and features related to the implementation of directives are usually considered complex by the survey participants.

4.6 AngularJS 2.0

To reveal the most expected features or improvements in ANGULARJS 2.0, we selected the following features that appeared in the mapping study: (1) the new API to define the main UI building blocks; (2) Zone.js (no more `$scope.$apply()`); (3) server rendering; (4) the new logging service, called diary.js; (5) new mobile features (e.g. support for touch event gestures); (6) the new template syntax; (7) the new change detection mechanism; (8) the new API for injecting dependencies; (9) integration with ECMAScript 6 (ES6) modules; (10) component router, which allows asynchronous loading.

Figure 7 indicates that the most expected feature is the integration with ES6 modules (68.3%). The second most expected feature is the new API to create UI components, which is expected because the current API is perceived by most developers as difficult to use and understand (see Section 4.5). The third most expected feature is the component router, which allows the dynamic loading of UI components, followed by the new change detection mechanism.

4.7 Key Findings and Implications

The main findings of our survey are as follows:

- Three characteristics of ANGULARJS excel by the value that developers give to them: the ability to create UI components by means of custom directives, the use of dependency injection, and the ease to set up two-way data binding. Therefore, JavaScript MVC framework builders can embrace these characteristics and improve them by offering a more simple interface to build reusable UI components (without exposing final users to internal concepts and decisions) and by using better mechanisms to detect changes in the model (i.e., mechanisms that reduce the number of details to be considered).
- The two problems that arise more frequently regarding the use of code in ANGULARJS templates are the emergence of silent failures and the difficulty to dump/debug the variables referenced in the HTML. New debug tools and techniques can then be developed to alleviate these problems faced by developers.
- The two main reasons for placing large amounts of logic in templates are the lack of experience in ANGULARJS and the design of the framework. This shows the importance of correctly training developers before they start to use ANGULARJS on complex applications. It also reveals an opportunity for framework builders to investigate new framework designs.
- The components that are more difficult to test in ANGULARJS are directives, mainly the ones using transclusion; the remaining components (i.e., controllers, services, filters, etc.) are mostly considered easy to test.

5. Threats to Validity

The first threat to validity relates to the execution of the mapping study. Due to the large amount of information on the Internet about ANGULARJS, it is possible that literature addressing more specific topics about the framework or presenting different points of view was not included.

Regarding the construction of the questionnaire, the main threat is the insertion of ambiguous and leading questions.

We made our best to avoid this problem by constantly reviewing and improving the proposed questions. Additionally, we ran a pilot survey to identify and correct this type of questions. In some questions, we also gave the participants the opportunity to respond with an answer different from the proposed ones by adding an "Other" option. Furthermore, with the exception of the background questions, no question was mandatory. Therefore, participants were not forced to provide answers when they did not want or when they were not sure about their answers.

There are also two threats related to the method used to retrieve the participants' e-mails. The first one is related with the match between the Stack Overflow profile and the GitHub profile of the participants. It is possible that a Stack Overflow user has been matched with a homonym user in GitHub (i.e., users who have exactly the same login name, but who are different people). Additionally, it is possible that the heuristic used to assess and rank the expertise of the selected developers does not reflect the reality.

There are also some aspects that limit the generalization of our results (external validity). First, the sample for the survey was selected only from the Stack Overflow community. Therefore, it is possible that the findings in this study do not apply to a different population. Moreover, constant and rapid changes in Web development environments, including new technologies and new versions of ANGULARJS, can lead to different results if the study is repeated in the future.

Finally, we have to mention threats related to human behavior. For instance, we can mention the ordering of the questions since one may provide context for the next one. Another threat is the attitude of the participants towards the topic of research. Their responses can introduce bias to make ANGULARJS appear in a positive or negative light. We can mention the case of one participant who declared that he is a contributor of the ANGULARJS project.

6. Related Work

Some works have been focusing on practitioners' use of known technologies. Dobing and Parsons (2006) conducted the first survey on how UML diagrams are used by practitioners and their clients. The authors gathered 182 responses from analysts with average experience of 4.7 years in UML. Class diagrams are being used regularly by the majority of the respondents, followed by Sequence and Use Case diagrams. Some of the reasons why a UML diagram is not used vary from "not well understood by analysts" to "insufficient value to justify the cost". In another study, Petre (2013) reported two years of interviews with practitioners. Most of them (35 out of 50) currently do not use UML at all, due to notation overhead, lack of context, etc. Both work highlight the complexity of UML. They also suggest that more tooling is needed for both newcomers and professionals in order to use the language more effectively.

Ocariza et al. (2015) proposed two types of inconsistencies that can be found in ANGULARJS applications: (i) when identifiers used in one layer are undefined in the lower layer; and (ii) when values assigned to a variable, or returned by a function, do not match their type in the view. According to the authors, both inconsistencies are not easily caught during development and might cause bugs. However, in our survey, only 39% of the respondents considered that *silent failures*, corresponding to identifier inconsistency, are real problems in their daily development. Moreover, 84.5% of the respondents considered two-way data binding, which relates to type consistency, as a valuable feature. Both results shed light over real problems developers face when they use ANGULARJS.

7. Conclusion

This paper reported an empirical study about different aspects of ANGULARJS based on opinions and experiences of developers. Our main contributions include the identification of the most appreciated features of ANGULARJS (e.g., custom interface components, dependency injection, and two-way data binding) and the most problematic aspects of the framework (e.g., performance degradation and implementation of directives). Future work includes an analysis of the results using statistical tests. Interviews with ANGULARJS developers can also contribute to strength our findings.

Acknowledgments

Our research is supported by FAPEMIG and CNPq. We also deeply thank the 460 developers who answered the survey.

References

B. Dobing and J. Parsons. How UML is used. *Commununications of the ACM*, 49(5):109–113, 2006.

H. M. Kienle. It's about time to take JavaScript (more) seriously. *IEEE Software*, 27(3):60–62, 2010.

H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, A. T. Nguyen, and T. N. Nguyen. Detection of embedded code smells in dynamic web applications. In *27th International Conference on Automated Software Engineering (ASE)*, pages 282–285, 2012.

F. Ocariza, K. Pattabiraman, and A. Mesbah. Detecting inconsistencies in JavaScript MVC applications. In *37th International Conference on Software Engineering (ICSE)*, pages 325–335, 2015.

M. Petre. UML in practice. In *35th International Conference on Software Engineering (ICSE)*, pages 722–731, 2013.

L. Silva, M. Ramos, M. T. Valente, N. Anquetil, and A. Bergel. Does Javascript software embrace classes? In *22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 73–82, 2015.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer, 2012.