AngularJS Performance Survey: Extended Technical Report

Miguel Ramos[1], Marco Tulio Valente[1], Ricardo Terra[2]

[1]Dept. of Computer Science, UFMG, Brazil
[2]Dept. of Computer Science, UFLA, Brazil
{miguel, mtov}@dcc.ufmg.br, terra@dcc.ufla.br

Abstract

This extended technical report provides detailed examples and technical discussions about our survey *under review* at IEEE Software Magazine. Therefore, this report describes an empirical study about different aspects of AngularJS performance, which uses as its main source of information qualitative data collected through a descriptive survey. The study design is presented in Section 1 and describes in detail the *Mapping Study* and the *Survey Design* that represent the two phases in which the study was conducted. In Section 2 we present the results of the survey and a brief discussion about them. Threats to validity are presented in Section 3. Finally, we conclude this report with some final remarks presented in Section 4.

1 Study Design

This study was conducted in two complementary phases. The first part is an exploratory empirical research that aims at obtaining knowledge about the state-of-the-art on practices, problems, and solutions regarding AngularJS performance. The second part of the study consists of a survey research whose construction is supported by the results delivered by the first research. The following subsections describe the relation between these studies and how they were designed and implemented.

1.1 Mapping Study

The main goal of this part of the study was to gather as much information as possible about AngularJS performance, including but not limited to the frequent problems faced

by developers who use the framework, the causes behind these problems and the possible solutions to them. The information collected in this study was used as a basis to build an exploratory survey that would help us to validate the information itself and provide useful findings based on the experience of different developers using the framework.

To achieve this goal we performed a review of the existing literature addressing this topic. We made an effort to find related literature in recognized digital libraries of Computer Science, but we could not find many titles. There are very few studies about JavaScript MVC frameworks such as AngularJS and no one focus on performance. For this reason, we decided to perform a mapping study (or scoping study) which is a research method less stringent than a systematic literature review [14]. A mapping study is more suitable for fields of study that have been less explored. Our efforts were focused on the Internet where the Google search engine was used as the main tool in the investigation.

We considered any informal literature contributing with information about the causes of performance problems in AngularJS, how often these problems occur, what are the practices suggested by experienced developers to avoid them, and what are the most common mistakes made by the less experienced ones that lead to these problems. We started our research using search queries such as "*AngularJS performance*" or "*performance problems in AngularJS*". The results listed in the first positions were examined manually to verify that the topic was inside the ones that we were interested on. The literature checked included mainly blogs, but we also checked forums, community groups, and Q&A sites such as StackOverflow.

During the revision of blog posts we applied a criteria to select or discard a text. We only maintained those posts that were well-written and in which the described background of the author included some experience in software development.

To simplify the analysis of the collected information we systematically logged in a document the most relevant quotations found in the different web sources. This document also has important data about the source such as the access URL, the date in which it was accessed, the name of the author, and the professional information of the author (when this information was available). An identifier was assigned to each source of information and each quotation registered in the created document uses these identifiers to reference the source from which it was taken.

We also used other ways to reach more literature in the web regarding AngularJS performance. While we were going through the literature already found, we also visited those sites referenced in the text, resulting in the application of a similar strategy used in systematic literature reviews called "snowballing" [11]. Furthermore, when a topic was frequently mentioned during the revisions, a new search query was used to find more information about that

specific topic. As an example, we can mention the search query "*ng-repeat performance*" that was built due to the frequency with which the `ng-repeat` directive was brought up in the sources reviewed. The data registered in the created document was then further analyzed to find some trends about problems, practices and opinions in the realm of AngularJS performance. Then common themes were identified and used as the basis for the construction of our survey.

1.2 Survey Design

As a result of the mapping study, we constructed a survey (built in Google Forms) with 31 questions that could be answered in approximately 10 minutes. The questionnaire is mainly made up of multiple choice questions and rating questions that use nominal or ordinal scales to measure the information provided by the respondents. Some of the multiple choice questions have only two possible options (yes or no). In a subgroup of these questions we used optional open-ended questions, which foster the respondents to extend the information provided by asking about a more specific point of the base question topic, so we could draw better conclusions in the analysis stage. When it was considered necessary, the multiple choice questions with more than two options had an additional option with statements such as "I am not sure" or "I have no idea" to avoid forcing a response when the respondents did not have enough knowledge to answer the question.

We used a five points ordered response scale for all of the rating questions. Labels were only put in the extremes of the scale to show the respondents the range of possible responses. For example, if we were asking for the frequency of a practice, the scale presented to the respondents included the labels "very rarely" and "very often" as options one and five, respectively.

At the end of the questionnaire we provided the respondents with a space in which they could leave an optional comment that they considered relevant for our research. We also gave the respondents the opportunity to specify their names and emails in case they wanted to receive the results of our research.

The questionnaire was divided into four sections that collected information about (1) the background of the users, (2) their development practices and perceptions, (3) general causes of AngularJS performance problems and (4) technical causes of these problems. The background section collected information about the familiarity of the developers with JavaScript and AngularJS, as well as the size of the largest project they had developed with these technologies. In the second section, we included questions related mostly to perceptions about AngularJS performance, and practices and common solutions to deal with this aspect of an

application. In the third and fourth parts of the questionnaire, the causes for applications with poor performance were addressed. The third part focused on general causes such as the experience of the developer or the architecture of the application, while the fourth part focused on technical causes such as incorrect use of `$scope.apply` or unnecessary use of two-way data binding. It is worth mentioning that due to the technical knowledge required to answer this last section, we explicitly asked the respondents not to answer the questions in which they felt that they did not have the knowledge required to provide an answer.

Since the survey targets developers familiar with AngularJS we used some well-established communities to look for the respondents. To promote the survey in these groups we created a standard post in which we explained the objective of the research and invited the members to participate in the survey. This post was then published in the selected communities.

After elaborating the first version of the questionnaire we ran a pilot in two national groups: a Google group[1] and a Google+ community[2]. We did it with the purpose of detecting possible improvements that could be made on the survey instrument. After the pilot round, we only found and corrected typographical errors.

The final version of the survey was promoted in other communities including the AngularJS community in Google+[3], the AngularJS Google group[4], and a community built in Reddit for AngularJS developers[5]. Furthermore, we also spread the survey link in Twitter accounts related to the AngularJS ecosystem. The survey was open during approximately three weeks (since early September 2015) to receive responses and in this period we obtained 87 responses. Since the unique corrections made to the first version of the questionnaire were of typographical type, we also considered the responses from the pilot phase to obtain a total of 95 responses.

Due to the nature of the methodology used to promote the survey, it is not possible to calculate accurately the response rate. Even when the number of members of each community is known, we can not determine how many of them visualized the survey post and how many of them are members of multiple communities.

The documents used in the mapping study and the survey questionnaire are publicly available in a companion repository.[6]

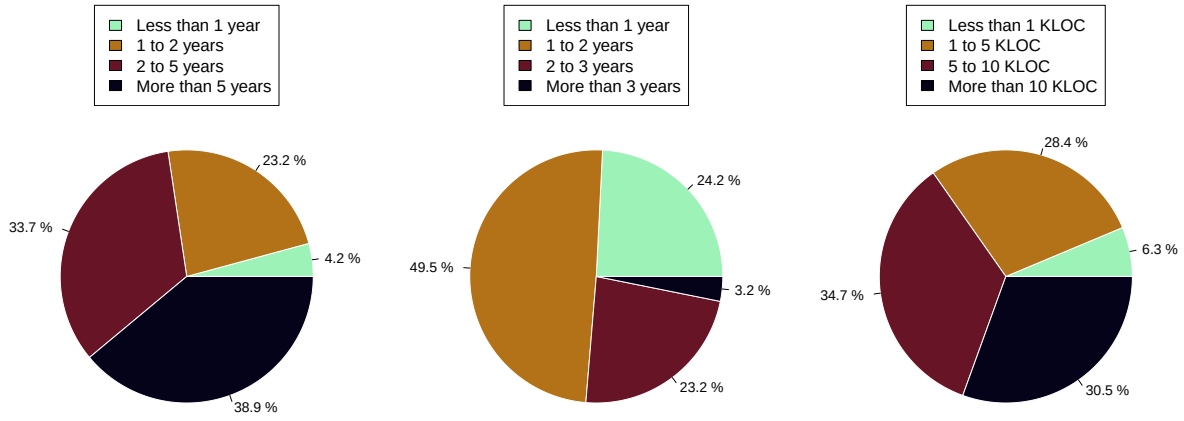
[1] *https://groups.google.com/forum/#!forum/angularjs-br*
[2] *https://plus.google.com/communities/110372004895086149874*
[3] *https://plus.google.com/communities/115368820700870330756*
[4] *https://groups.google.com/forum/#!forum/angular*
[5] *https://www.reddit.com/r/angularjs/*
[6] *https://github.com/aserg-ufmg/angularjs-performance-survey*

2 Results

In this section, we present the results of the survey and the analysis of the responses. The questions are grouped by topic to facilitate their interpretation. The first group provides information about the respondents background and it is the base for making comparisons through the rest of the survey.

2.1 Background

Figure 1 presents the distribution of the background answers. These answers give us a picture about the experience of the participants in JavaScript and in AngularJS. As we can see (Figure 1(a)), 72.6% of the respondents reported an experience of at least two years in JavaScript and 38.9% reported to have been working more than five years with the language. Figure 1(b) shows that 24.2% of the developers have less than one year of experience in AngularJS. The other part of the developers (75.8%) is divided as follows: 72.7% declared to have an experience from one to three years in the framework and only 3.2% to have an experience for more than 3 years. This is probably due to the date of creation of AngularJS (2009) and the date in which it became popular (2013).



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

Figure 1: Respondents background

In order to complement the background information, we asked developers about the largest application they have developed using AngularJS. The results are shown in Figure 1(c). Only 6.3% of the developers reported that their largest application has less than 1 KLOC. On the other hand, 30.5% declared to have developed applications with more than

10 KLOC. The other 63.1% said that their largest application has between 1 KLOC and 10 KLOC.

Therefore, most of the respondents have a considerable experience in JavaScript while a more distributed experience is observed in AngularJS. Nevertheless, half of the participants have an experience in the framework that ranges from one to two years. In relation to the size of the applications implemented by them, we have a similar proportion of developers writing small, medium, and large applications with AngularJS.

2.2 How Developers Improve AngularJS Performance

As a result of the mapping study we found some trends in the practices followed by developers to improve the performance of their applications. Some of these practices suggest that developers need to know implementation details of the framework, substitute or improve some parts of the framework with other components, or even modify the source code of the framework. As an example we can see the following descriptions of the components created for this purpose ([3]):

sly-repeat is our variant of ng-repeat, which hides extra DOM elements rather than destroying them. sly-evaluate-only-when prevents inner change watchers from executing unless... And sly-prevent-evaluation-when-hidden prevents the inner repeat clause from executing until the mouse moves over this line and the div is displayed.

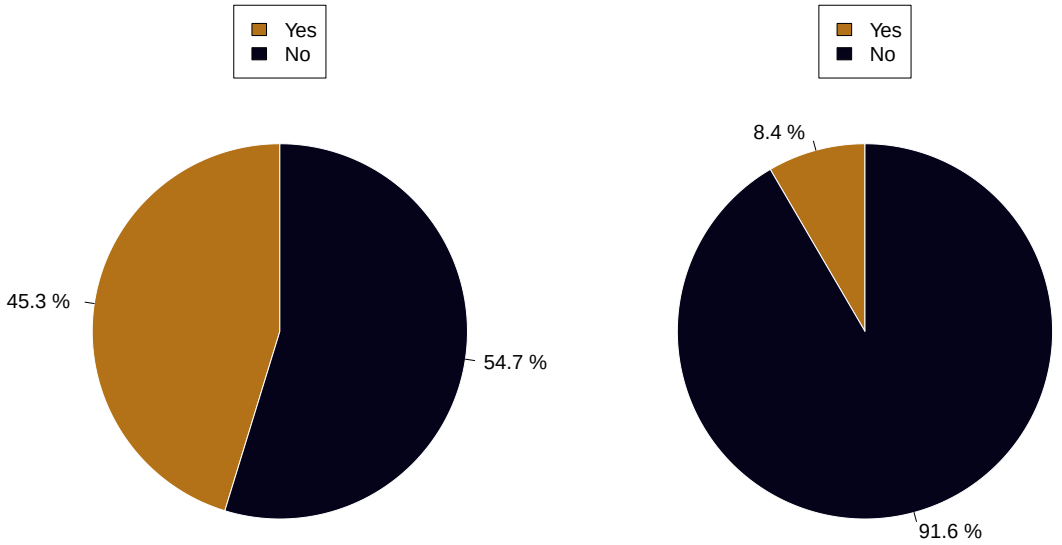
At the end of the post it is also said that:

We overrode the Scope's $watch method to intercept watcher registration, and then had to do some careful manipulation of Scope's instance variables to control which watchers are evaluated during a $digest.

To find how popular these practices are among AngularJS developers, we created a set of questions whose results are discussed below. These practices do not include actions such as small refactoring operations.

Figure 2 presents the distribution of the responses about interactions with the source code of AngularJS (Questions 7 and 8 of the survey questionnaire). We can see how often developers have inspected the AngularJS source code looking for performance improvements and how often they even modified the source code for this purpose. Figure 2(a) shows that almost half (45.3%) of the developers have at least once inspected the AngularJS source

code to figure out how to improve the performance of their applications. This is a clear evidence that developers seek to understand the internals of AngularJS when fixing performance problems. On the other hand, only 8.4% (8 developers) reported to have made changes in the AngularJS source code to deal with performance problems (Figure 2(b)). The reason is probably because this is a bad practice with serious implications in the maintenance and evolution of the software product.



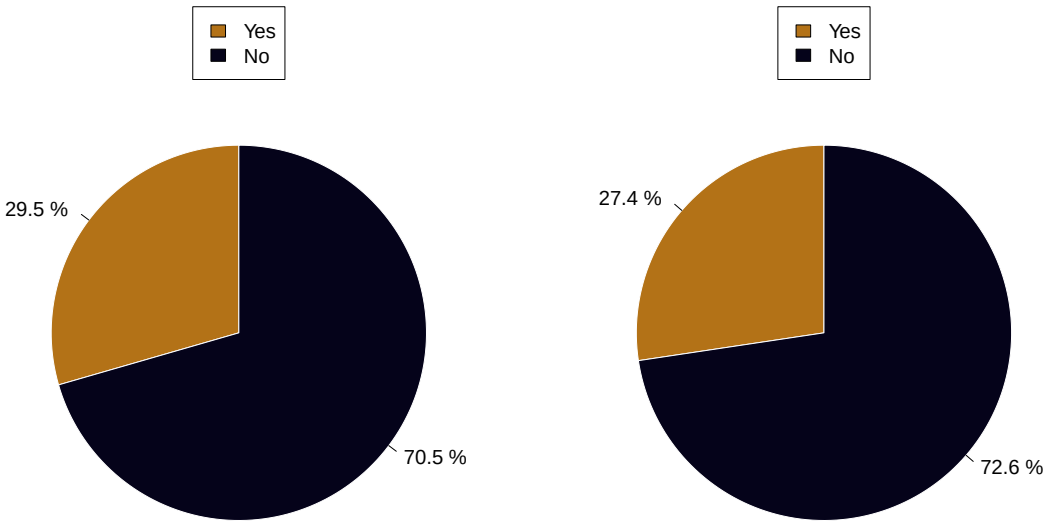
(a) Have you ever inspected AngularJS source code to improve performance?

(b) Have you ever changed AngularJS source code to improve performance?

Figure 2: Interactions with AngularJS source code

Another trend found about how developers improve the performance of their applications in AngularJS regards the substitution of features provided by the framework with custom components. These components are created by the project owner itself or by third parties. Figure 3 shows the percentage of developers who have created or used third-party components to improve the performance of their applications (questions 9 and 11 of the survey questionnaire). As we can see 29.5% of the respondents reported that they have used third-party components to improve performance and 27.4% reported that they have created a custom component for this purpose.

To reveal the roles of these components, we set two open-ended questions asking (1) the name of the third-party components used and (2) a description of the custom components created (questions 10 and 12 of the survey questionnaire). We analyzed these answers qualitatively. For the third-party components we used the names to search for the purpose of their



(a) Have you ever used a third-party component to improve the performance of an AngularJS application?

(b) Have you ever created a custom component to improve the performance of an AngularJS application?

Figure 3: Developers using and creating components to improve application performance

creation on the Internet. We found that the two most popular reasons for which these components have been used are: (1) to be able to bind once when this feature was not available in AngularJS (before version 1.3) and (2) to substitute or improve the standard `ng-repeat` directive. In the first category, the only component mentioned was `bind-once` while in the second category we found names such as `angular-virtual-scroll`, `infinite-scroll`, `ng-table` and `collection-repeat` from the Ionic Framework.

When analyzing the descriptions of the created custom components we found a clear trend in their purpose. Most of them deal with large amounts of data that need to be rendered as tables or long lists. This result is consistent with our mapping study, when we found a considerable number of developers facing problems with the `ng-repeat` directive when handling large collections of data, and developers and third-party modules providing solutions for this problem. As examples, we can mention some posts found in StackOverflow with titles like *"Angular performance in ng-repeat"* [9] and *"How to improve performance of ngRepeat over a huge dataset (angular.js)?"* [12].

The last result that we present in this section has to do with performance in mobile devices. In some of the reviewed blogs, we found developers saying that performance problems in AngularJS got worse in mobile development [6, 5]. Therefore, we asked developers if they

had ever made refactorings in their applications due to performance issues in mobile devices (question 13 of the survey questionnaire). Figure 4 shows that 34 developers (35.8%) answered positively to this question.

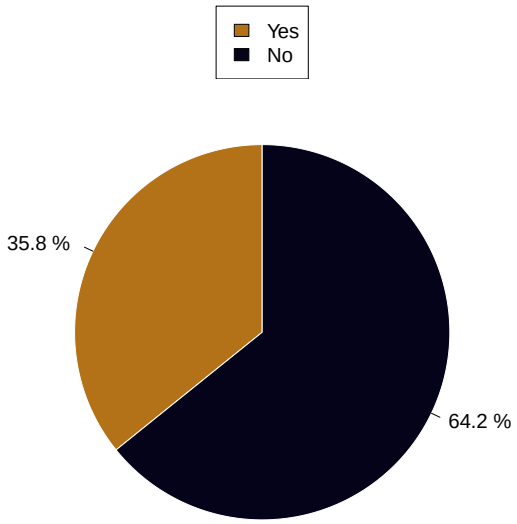


Figure 4: Developers who refactored their AngularJS applications specifically to improve performance on mobile devices

2.3 General Causes of Performance Problems in AngularJS

There are different factors that may lead to low performance in AngularJS. At the highest level, the causes of performance issues can be things like the experience of the developer or the architecture of the framework. We decided to include some general questions to find out what are the most probable causes based on the experience and opinions of the developers. We chose four possible reasons: (a) the lack of experience in the language (JavaScript), (b) the lack of experience in the framework (AngularJS), (c) applications following wrong architectures and (d) the internal architecture of AngularJS. Figure 5 presents the answers of the developers on how often these causes are responsible for applications with performance problems (questions 18 to 21 of the survey questionnaire). To make this evaluation, they rated each cause in a scale from one (very rarely) to five (very often).

The first group of bars (from left to right) in Figure 5 represents the result for the first cause: lack of experience in JavaScript. The most common answer was the neutral option (*three*) with 28.4%. The number of answers for the options *two* and *four* was the same (23

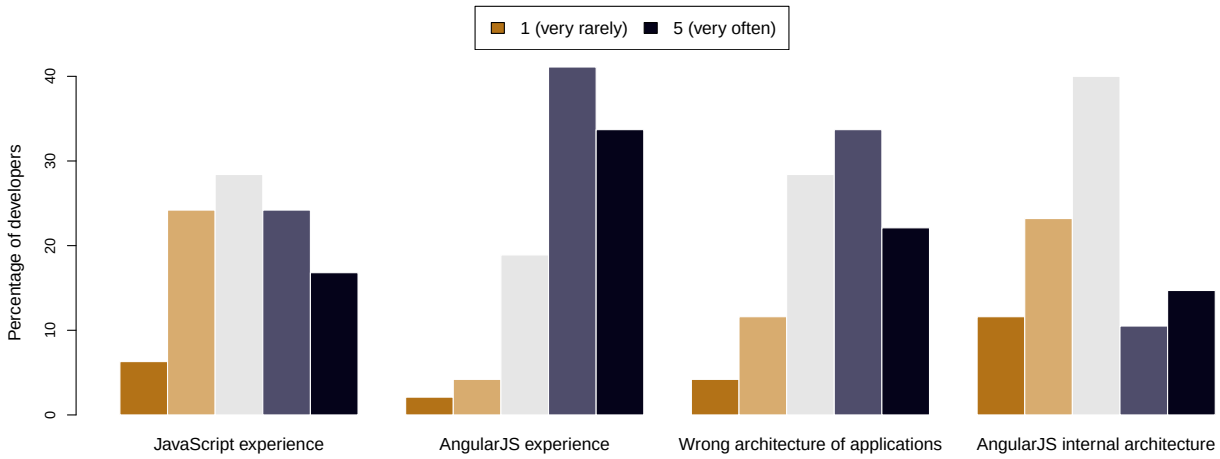


Figure 5: Evaluation of general causes of performance issues in AngularJS

answers), and the difference between the extreme options (10.5%) is not enough to draw a conclusion.

On the other hand, the second group of bars shows a more clear tendency in the responses. The answers of 74.8% of the respondents correspond to one of the highest options in the scale (*four* or *five*) and only 6.3% to the lowest option (*one* or *two*). Therefore, according to the respondents, the lack of experience in AngularJS is a frequent cause of developers building applications with performance problems when using this framework.

The third group of bars, representing the cause "applications following a wrong architecture", also shows a trend to the high part of the scale; however, this is less strong than in the previous case. We can see that 55.8% of the developers gave an answer of *four* or *five* and that 15.8% gave an answer of *one* or *two*. These results reveal that most developers also agree that inadequate architectures are frequently a cause of poor performance. In other words, the problem is not related with the framework but with the ability of the developers to define architectures that allow high performance in their applications.

In the last group of bars we present the opinion of the developers about how often the internal architecture of AngularJS can be blamed for poor performance. The first result worth noting is that this is the group in which the neutral option is the most popular answer with 40%. The second is that there is not a strong trend in either side of the scale (34.8% of the answers are in the low part of the scale and 25.2% in the high part). In summary, we can not conclude that the internal architecture of AngularJS is considered by the developers as a cause of performance problems.

The most clear conclusion that we can draw from this section is that many developers agree that the lack of experience in AngularJS is a very likely reason for which their peers build applications with poor performance. However, this can also be seen as an expected result. It is almost certain that developers who know better the tools with which they work will develop applications with better quality. But, how much do they need to know about AngularJS to avoid performance degradations in their applications? To answer this question in Section 2.4 we investigate the relation between poor performance and technical aspects of the framework.

As a final comment, we must state that the findings presented in this section are more based in opinions and not with the experience or practice of the developers. Therefore, bias may have a major effect in the results due to the population that was selected to answer the survey.

2.4 Technical Causes of Performance Problems

During the mapping study we also found several recommendations (based on technical knowledge) about practices that should be followed (or avoided) to deal with performance problems in AngularJS [8, 10, 7]. From this study, we extracted 13 practices than can lead to performance degradation. A brief description of each practice is presented below.

1. *Unnecessary use of two-way data binding:* Two-way data binding is the solution offered by AngularJS to keep in synchrony the data between the model and the view components. However, every time that a binding of this type is used a new `$watch` expression is added to be checked for changes during the digest cycle. Therefore, the higher the number of bindings of this type, the greater the cost of the digest cycle. The release 1.3 of AngularJS introduces a way to implement one-time data binding, which consists in expressions that are no longer checked for changes once a stable value is assigned to them. Therefore, this feature reduces the number of expressions being watched and makes the digest cycle faster. Unfortunately, sometimes two-way data binding is used when one-time data binding is enough leading to an unnecessary degradation of the application performance.

2. *Use of a wrong watching strategy:* During the digest cycle, the change detection process is performed. The current values of the expressions being watched are compared to the old ones. This comparison can be done in three different ways: by reference, by collection, or by value. If an object is being checked, for example, a comparison by reference only is able to detect a change when the whole object is replaced by another.

When a comparison is made by collection or by value changes made in the nested values of the object can be detected. However, a comparison of this type is expensive since a full traversal of the nested data structure is needed on each digest, and a full copy of it needs to be held in memory [1]. If the developer misuses the most expensive watching strategies, the performance of the whole application can be negatively affected.

3. *Watching complex functions:* Each scope in an AngularJS application allows the developer to register a function whose returned value will be watched. This registration can be done through the `$watch` method. In each digest cycle the function is executed and the returned value is compared with the value returned by the function in its last execution. If the function is highly complex, its execution will take longer, as well as the digest cycle. If many complex functions are watched, the performance of the application can drop considerably.

4. *Use of filters in templates instead of in JavaScript code:* In AngularJS, filters are functions applied to the value of expressions to format data displayed to the user. They can be used directly in templates with an especial syntax (`{expression | filter}`) or they can be used in controllers, services and directives using JavaScript code. However, when a filter is used in a view template, it is evaluated on every digest cycle, which can be very expensive specially for large data structures. On the other hand, when a filter is used in a controller, it is possible to call the filter only when needed avoiding its execution on each digest cycle.

5. *Use of mouse events directives:* AngularJS provides developers with directives such as `ngClick`, `ngMouseenter`, `ngMouseleave`, etc., that allow the tracking of mouse events. Since the functions that handle these events can change the model data, each time that a user interaction associated with the directive is performed, the digest cycle is triggered for change detection.

6. *Use of a wrong conditional-display strategy:* There are two different types of directives that can be used to display DOM elements depending on a condition. The first type of directive (e.g., `ngHide` and `ngShow`) hides and shows elements by using CSS classes, without removing them. The second type of directive (e.g, `ngIf` and `ngSwitch`) actually creates or destroys DOM elements depending on a condition. The main advantage of the last strategy is that along with the DOM element are also destroyed all the event handlers and bindings (and watches) associated to it. However, creating and destroying elements are expensive operations. Therefore, the type of directive used should be carefully selected depending on the situation.

7. *Inappropriate use of `$scope.$apply()`:* `$scope.$apply` is the method used by AngularJS to trigger the full digest cycle. It is implicitly called by the framework when there is a possibility of mutation in the model data. However, the operations responsible for these mutations can also occur in places where AngularJS has no control (e.g., in a third party library). For this reason, AngularJS allows the developers to explicitly call the `$scope.$apply` method. An abuse of this method may result in multiple triggers of the digest cycle.

8. *Runtime configurations and flags:* By default AngularJS attaches information (i.e., CSS classes) about bindings and scopes to the DOM elements so that debugging or testing tools can work. However, when the application is in production, it is recommended to disable this feature.

9. *Not cleaning up the application:* This problem arises when some parts of the application use computational resources even when they are no longer needed. This can occur, for example, when there are handlers that will never be triggered because the event emitter has been destroyed. Therefore, it is important to unbind handlers, remove unnecessary watches, and stop asynchronous operations, etc., when these actions are not done automatically. Most components of an AngularJS application have a destroy event that can be listened to destroy other components that depend on them, which can avoid losses in performance.

10. *Use of `ngRepeat` with long lists:* The `ngRepeat` directive is used to create DOM elements for each item in a collection. It is used to render lists and maintain the view updated when their items change. A new scope and a binding is created for each item to track the possible changes that can occur in the collection. When the collection is too large, these bindings can affect the digest cycle and therefore the performance of the application.

11. *Use of `ngRepeat` without "track by":* As mentioned before, the creation and deletion of DOM elements are expensive operations. When the `ngRepeat` directive is used AngularJS assigns a unique hash key to each item of the collection to make an association with its DOM element and avoid those expensive operations. However, when the collection is refreshed from the server this assignment is lost and all the elements must be re-rendered in the view. To solve this problem, the *track by* expression can be used with the `ngRepeat` directive to specify a unique key to each item to make the identification. If, for example, the identifiers of the objects loaded from the server are

used for this purpose, even when the collection is reloaded it is possible to recognize the items that have been rendered before.

12. *Nested directives in `ngRepeat`:* When the items of the collection used in the `ngRepeat` directive are represented by templates that use more directives creating bindings (e.g., `ngShow`), it is easy to create a lot of watches in the digest cycle.

13. *Unconscious triggering of the full digest cycle:* To guarantee the synchronization between the view and the model, AngularJS automatically triggers the digest cycle when operations that can change the data in the model are executed. Some methods that trigger the full digest cycle are `$http`, `$resource`, `$timeout`, `$interval`, `$scope.$apply`, etc. The lack of knowledge of this fact can lead to architect the application in a way in which the digest cycle is triggered multiple times.

To find out the relevance of the aforementioned situations, we asked the developers to rate how often they are responsible for negative effects in the performance of an AngularJS application (questions 22 to 24 of the survey questionnaire). For this purpose, the participants were given a scale for each item in which the lowest option (*one*) was labeled with "very rarely" and the highest option (*five*) was labeled with "very often".

Figure 6 presents the distribution of the responses for each item. The first observation is that there is no bar reaching 100% of the population. This observation may be due to the request that we made to the participants to avoid answering questions in which they did not have the necessary knowledge to provide an answer. As we can see, the cause with the lowest number of responses was the one addressing the configuration of the application for production mode (70.5% of the population gave an answer). On the other hand, the causes that received the largest number of responses were the ones about watching complex functions and using unnecessary two-way data binding, in which 87.4% of the population participated.

In Figure 6, the last three bars show the clearest trend towards the right end of the scale, which means that they affect the performance of AngularJS applications with more frequency. The unnecessary use of two-way data binding was signaled by 31.3% of the developers who gave an answer as a very frequent cause of performance degradation. Another 36.1% rated this cause with *four* in the scale, which represents 67.4% of the responses in the highest part of the scale. Furthermore, only 15.6% of the responses are located in the lowest options of the scale for this cause (7.2% in option *one* and 8.4% in option *two*).

This result may be related to two questions from the first section of the questionnaire. In the first one we asked the participants to select the version of AngularJS with the major

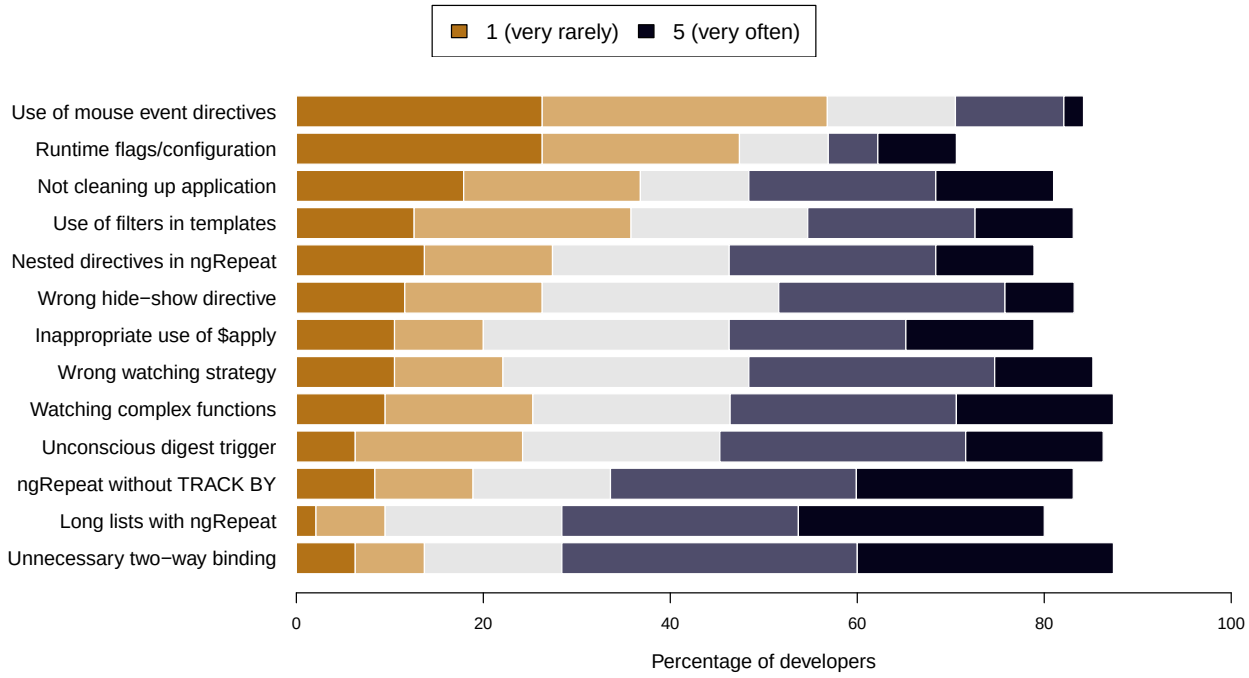


Figure 6: Evaluation of technical causes of performance issues in AngularJS

performance improvements. Removing the participants who answered "I am not sure" for this question (40% of the responses), 63.2% indicated version 1.3. A possible explanation for this result could be that in that version the option to use one-time data binding was introduced in AngularJS. The second question was about refactoring. We asked the developers if they had ever made a refactoring to replace two-way with one-time data binding to improve its performance. From the whole sample, 72.6% of the participants declared to have made at least once a refactoring of this type. This is another clear indicative of the high impact that this design decision has in the performance of an application.

The other two bars with a considerable amount of responses in the high part of the scale are associated to the use of the `ngRepeat` directive. Regarding the use of long lists with the `ngRepeat` directive, 31.6% of the developers marked the option *four* and 32.9% marked the option *five* in the scale. In total, 64.5% of the responses are in the highest part of the scale for this cause. Additionally, the responses in the lowest part of the scale (options *one* or *two*) represent only 11.8% of the total. These results make evident that most developers think that performance can be deteriorated by rendering long collections with the `ngRepeat` directive. This is not a surprising result given that, as was mentioned in Section 2.2, this is a consistent topic addressed in the literature found in our mapping study and it is one of the problems that

the AngularJS community has been trying to solve through third-party components. The problem with this directive is that it can be easily used to create complex DOM elements for each item in a collection creating at the same time several bindings (watches) that slow down the completion of the digest cycle. For this reason, most of the solutions proposed to solve this problem include the reduction of the displayed list by using strategies such as pagination or infinite scroll [2, 4].

Another practice, related with the rendering of long lists, that was also highly rated as a frequent cause of performance degradation is the use of the `ngRepeat` directive without the *track by* expression. The ratings for this cause are shown in the third bar of Figure 6. We can see that more than half of the developers selected an option from the high part of the scale for this cause. To be more specific, 31.6% selected the option *four* and 27.8% selected the option *five*. The neutral option was chosen by 17.7% of the developers who gave an answer and the lower options were selected by 22.8% of the participants. This reveals how much attention must be put in the architecture of AngularJS applications when working with large amounts of data representing items that need to be rendered and kept updated during the application execution. We can take as an example the use of a global identifier that allows the direct association of items in the server with the elements rendered in the client. In this case, expensive operations such as creation and deletion of DOM elements can be avoided by using the *track by* expression with the `ngRepeat` directive.

Another point worth mentioning about the results presented in Figure 6 is the classification given to the "use of mouse events directives" and the "use of wrong configurations for production" as unusual causes of performance degradation. For both scenarios more than 67% of the developers used options *one* or *two* to rate the frequency with which they were the problem behind the poor performance of an AngularJS application. In the case of mouse-events directives the reason for these results may be that using these directives by themselves does not affect the performance of the application. The problem appears only when they are spread all over the application creating multiple potential triggers of the entire digest cycle.

Disabling debug information is also an action that is not seen by the respondents as an important performance booster. The addition of information (scopes and CSS classes) to the DOM elements is mostly perceived while the application is being loaded for the first time. This optimization has little or no effect in the digest cycle and for this reason can be considered by the respondents as having a minor improvement in the performance. During our mapping study we also found developers saying that no improvements in performance are perceived after disabling debug data in production [13].

Finally, from Figure 6 we can declare that almost all of the technical causes considered for the construction of this survey are recognized by the developers as relevant in the deterioration of performance. With the exception of the two causes represented by the two bars in the top, at least 34.2% of the developers rated all the proposed causes with *four* or *five*. Maybe this is because all of them are related with three important programming practices that have to be kept into account to avoid problems of performance: (1) keep the digest cycle as light as possible by avoiding unnecessary and complex watches, (2) avoid multiple triggers of the digest cycle and (3) reduce as much as possible the number of expensive interactions with the DOM API.

3 Threats to Validity

The first threat to validity is related to our mapping study since it is not possible to guarantee that the sources reviewed include all of the available information about performance in AngularJS. In other words, information not collected during our mapping study could modify the construction of our survey leading to different results.

In the construction of a survey there is always the risk of having ambiguous or leading questions. However, we made our best to avoid this type of questions by using multiple iterations of small reviews and corrections of the proposed questions. We also used the pilot phase to refine the wording of the questions. Additionally, it is important to note that we did not use mandatory questions whenever we thought that some kind of special knowledge was necessary to avoid forcing responses.

There are also some aspects that limit the generalization of our results (external validity). First, the subject population was selected from specif on-line communities and it is not possible to make sure that the findings also apply to populations not participating in these communities. Additionally, due to the constant and rapid change in the Web development environment, it is possible to obtain different results if the study is repeated in the future. As an example, we can mention that while this study was being conducted a new release of the framework (1.5) was being developed and also a new version (2.0) for which the results could no longer apply.

Finally, we have to mention the threats to validity related with social aspects. For example, the ordering of the questions could affect the answers of the respondents. The responses can also be affected, either positively or negatively, by the attitudes of the developers to the research topic or their tendency to look better when they are interrogated. However, this last threat may be alleviated by the anonymous character of the responses.

4 Final Remarks

This report reported an empirical study about AngularJS performance supported by a survey. We started by describing the design of the study, which was divided in two phases. The first phase was a *Mapping Study* (Section 1.1) in which we gathered information about the state-of-the-art of AngularJS performance by reviewing informal literature that was found mainly on the Internet. The second phase was the design and execution of a survey that we described in details in Section 1.2. More specifically, the topics addressed in the survey were (a) background of the users, (b) development practices and perceptions regarding AngularJS performance, (c) general causes of poor performance in AngularJS applications, and (d) technical causes of this problem.

The results of the survey were presented in Section 2 where we discovered, for example, that many developers (45.3%) reported to have inspected the code of the framework to fix performance issues. We also concluded that the most frequent causes (from a general point of view) of poor performance are the lack of knowledge in AngularJS and applications following a wrong architecture. In Section 2.4, we analyzed the technical errors made by developers that can lead to performance degradation. Based on the results, the most frequent causes of performance degradation in AngularJS applications are related with the unnecessary use of two-way data binding and the use of the `ngRepeat` directive with long lists or without the *track by* expression. On the other hand, we also concluded that using mouse-events directives or not using the adequate configurations for production are not considered as major threats to performance by the developers. Finally, in Section 3 we discussed the threats to validity of this study which are mainly concerned with the sample selected, the social behavior presented by the respondents, and the construction of the survey instrument.

References

[1] Scope $watch performance considerations. *https://docs.angularjs.org/guide/scope*, 2015. [Online; accessed October-2016].

[2] Chandermani Arora. AngularJS performance. *https://www.packtpub.com/books/content/angularjs-performance*, 2015. [Online; accessed September-2016].

[3] Steven Czerwinksi. Optimizing AngularJS: 1200ms to 35ms. *http://blog.scalyr.com/2013/10/angularjs-1200ms-to-35ms/*, 2013. [Online; accessed June-2015].

[4] Sebastian Fröstl. AngularJS performance tuning for long lists. *http://tech.small-improvements.com/2013/09/10/angularjs-performance-with-large-lists/*, 2013. [Online; accessed September-2016].

[5] Peter Koch. The problem with Angular. *http://www.quirksmode.org/blog/archives/2015/01/the_problem_wit.html*, 2015. [Online; accessed September-2016].

[6] Egor Koshelko. Why you should not use AngularJS. *https://medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99*, 2015. [Online; accessed September-2016].

[7] Alex Kras. 11 tips to improve AngularJS performance. *http://www.alexkras.com/11-tips-to-improve-angularjs-performance/*, 2015. [Online; accessed July-2015].

[8] Radek Markiewicz. Angular non-trivial performance hints. *http://codetunes.com/2015/angular-non-trivial-performance-hints/*, 2015. [Online; accessed July-2015].

[9] Kashif Mustafa. Angular performance in ng-repeat. *http://stackoverflow.com/q/22937504/5244036*, 2014. [Online; accessed September-2016].

[10] Ben Nadel. How often do filters execute in AngularJS. *http://www.bennadel.com/blog/2489-how-often-do-filters-execute-in-angularjs.htm*, 2013. [Online; accessed June-2015].

[11] Mats Skoglund and Per Runeson. Reference-based search strategies in systematic reviews. In *13th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 31–40, 2009.

[12] How to improve performance of ngRepeat over a huge dataset (angular.js)? *http://stackoverflow.com/q/17348058/5244036*, 2013. [Online; accessed September-2016].

[13] AngularJS disabling debug data in production. *http://stackoverflow.com/q/32967438*, 2015. [Online; accessed September-2016].

[14] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer, 2012.