

AngularJS Performance

A Survey Study

Miguel Ramos and Marco Tulio Valente, Federal University of Minas Gerais

Ricardo Terra, Federal University of Lavras

// Researchers surveyed 95 professional developers regarding the performance problems of AngularJS applications. They identified the common practices the developers followed to avoid the problems, and they determined the problems' general and technical causes. //

52,718 for AngularJS, compared to 620 and 16,938 for Ember.js and 289 and 25,615 for Backbone.js), as collected in October 2016.

Websites that use AngularJS include weather.com, forbes.com, and intel.com (according to libscore.com, a service that collects stats on JavaScript library use).

Despite the increasing interest in AngularJS, few studies have examined the performance of applications constructed with it, including the recurrent performance problems that AngularJS users face and the possible causes and solutions. To offer some best practices to deal with AngularJS performance problems, we report here the results of a survey about these problems. The survey also revealed the AngularJS design decisions and features that are usually considered the problems' causes.

Survey Design

To get information about the main challenges and problems developers face regarding AngularJS performance, we used a mapping study. Because such studies are more flexible than systematic literature reviews, they're recommended for studying emergent fields or technologies,³ such as AngularJS. Because there's limited formal literature on AngularJS, we focused on blogs, forums, and Q&A sites (for example, Stack Overflow).

To collect the first documents of interest, we used Google search queries such as "AngularJS performance" or "performance problems in AngularJS." We verified that the retrieved documents discussed AngularJS performance. For frequently mentioned topics, we performed new searches to find more

JAVASCRIPT IS A fundamental piece of modern web apps that's used to construct a variety of systems, including web apps with sophisticated user interfaces. As a result, we're witnessing the birth of new technologies and tools (including JavaScript libraries and frameworks) to solve common problems in such apps. Specifically, a new family of JavaScript frameworks has emerged, following the model-view-controller architecture pattern (or variations of it). Examples include AngularJS, Backbone.js, and Ember.js.

Among them, the most popular is AngularJS,^{1,2} which Google built and maintains. This popularity is evident when we compare

- the number of Google searches (AngularJS is the most queried framework since 2013; see Figure 1);
- the number of questions in Stack Overflow (AngularJS has the most questions per month since 2013; see Figure 2); and
- the numbers of contributors and stars at GitHub (1,530 and

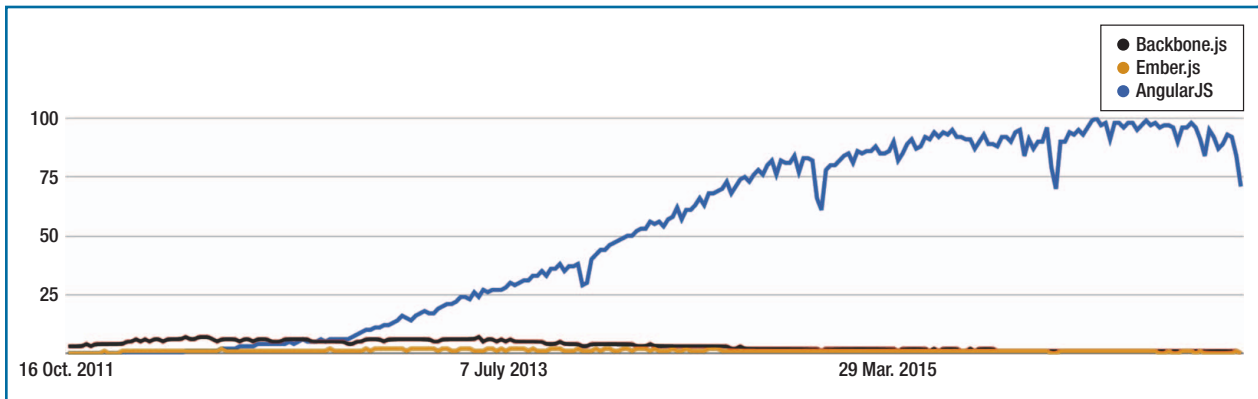


FIGURE 1. Google searches for AngularJS, Backbone.js, and Ember.js. AngularJS is the most queried framework.

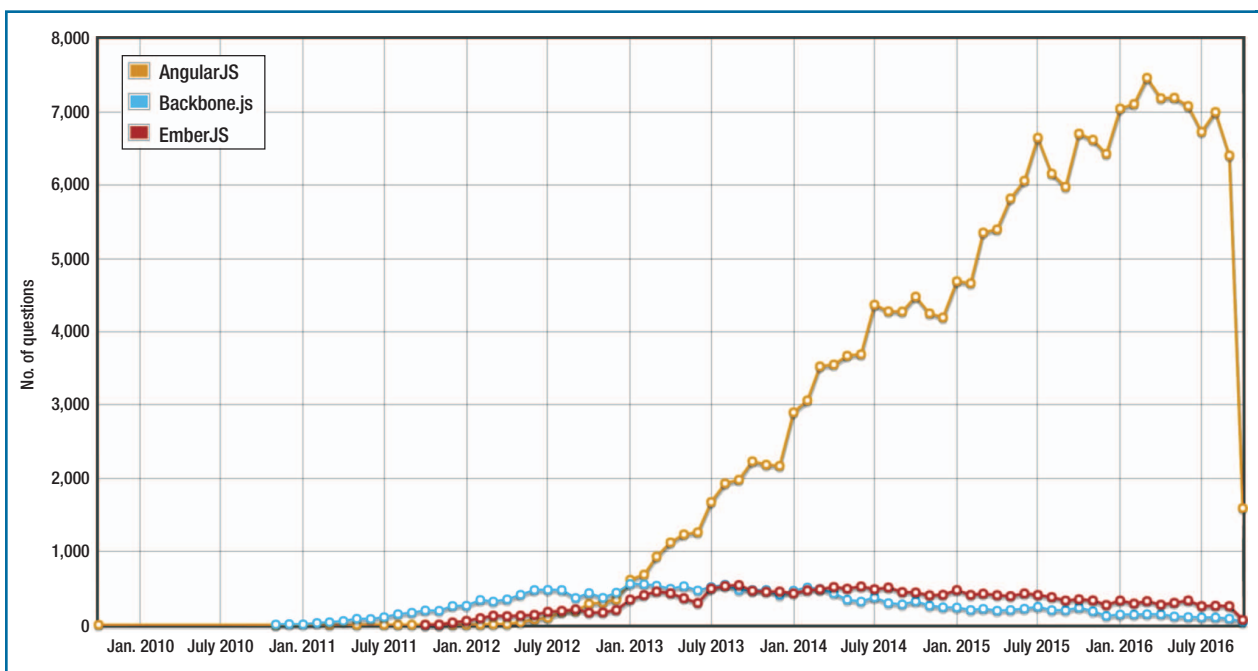


FIGURE 2. The number of Stack Overflow questions per week for AngularJS, Backbone.js, and Ember.js. AngularJS has the most questions per month.

information. For example, we queried for “ng-repeat performance” because discussions about the **ng-repeat** directive appeared often in the initial set of reviewed documents. We collected 25 total documents (3 AngularJS documents, 14 blog posts, and 8 Stack Overflow questions). We reviewed each document to identify

common trends and discussions, which we used in our survey design.

The survey had 31 questions divided into four sections: background, practices and perceptions regarding AngularJS performance, the performance problems’ general causes, and the problems’ technical causes. The questions regarding the technical

causes weren’t mandatory because they might have required knowledge of the framework’s advanced features. So, the respondents didn’t have answer to those questions if they hadn’t mastered a specific feature that caused performance issues.

We promoted the survey in popular AngularJS communities

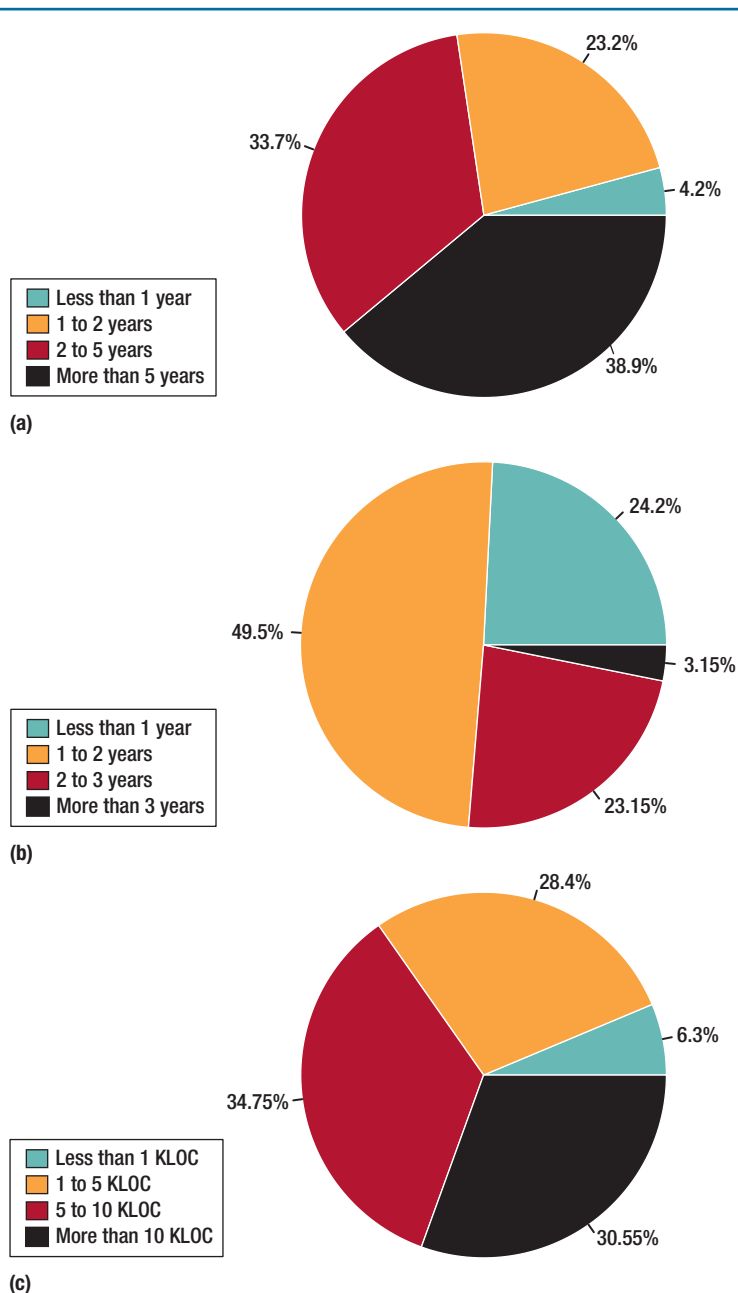


FIGURE 3. The survey respondents' backgrounds. (a) JavaScript experience. (b) AngularJS experience. (c) The size of the largest application the respondents had worked on. Most of the respondents weren't novice AngularJS developers.

and forums, including the AngularJS Google group (groups.google.com/forum/#!forum/angular), the AngularJS community in Google+

(plus.google.com/communities/115368820700870330756), and a Reddit community (www.reddit.com/r/angularjs). The survey was open

for three weeks (starting in early September 2015); we obtained 95 responses. The documents used in the mapping study and survey questionnaire are at github.com/aserg-ufmg/angularjs-performance-survey.

The Results

As Figures 3a and 3b show, 72.60 percent of the survey respondents had at least two years' experience in JavaScript, and 75.80 percent had at least one year's experience in AngularJS. Only 3.15 percent had more than three years' experience in AngularJS because the framework started becoming popular around 2013 (although its first release was in 2009). Only 6.30 percent reported that their largest AngularJS application had less than 1 KLOC (see Figure 3c). So, we conclude that most of the respondents weren't novice AngularJS developers.

How Developers Improved AngularJS Performance

To improve AngularJS apps' performance,

- 45.3 percent of the respondents inspected the AngularJS source code at least once,
- 8.4 percent changed the AngularJS source code,
- 29.5 percent used third-party components, and
- 27.4 percent created a custom component.

The two most common reasons for using custom or third-party components were to

- support **bind-once** when this feature wasn't available (before version 1.3) and
- replace or improve **ng-repeat**.

In the first case, the only component the respondents mentioned

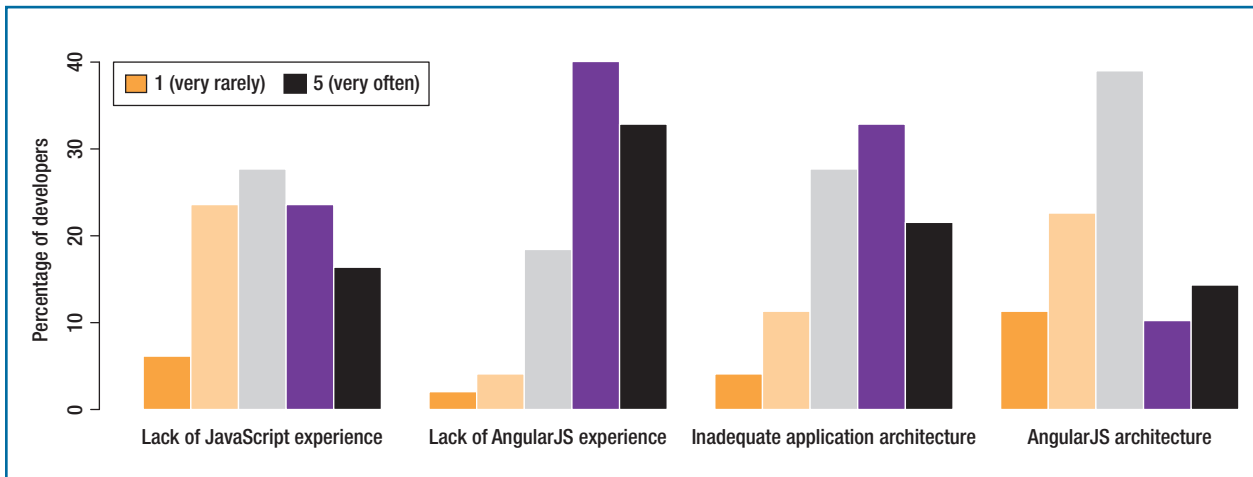


FIGURE 4. The occurrence of the four general causes of performance problems in AngularJS. The respondents ranked the occurrence on a scale of 1 (very rarely) to 5 (very often). We considered a ranking of 3 to be neutral.

was `bind-once`; in the second case, they mentioned `angular-virtual-scroll`, `infinite-scroll` (which is no longer maintained), `ng-table`, and `collection-repeat` from the Ionic framework. These components deal mostly with large amounts of data in tables or lists, which confirms a topic that emerged in the mapping study.^{4,5}

Our mapping study discovered documents reporting that the AngularJS performance problems were more critical in mobile phones.^{6,7} When we asked the developers about this, 35.8 percent of them answered that they refactored their code at least once to improve performance on mobile devices.

The Problems' General Causes

Figure 4 shows how the respondents ranked the four possible general reasons for performance problems, using a scale of 1 (very rarely) to 5 (very often).

Lack of JavaScript experience. The most common ranking was 3 (neutral),

by 28.4 percent of the respondents. The rankings of 2 and 4 both received 24.2 percent, and the difference between the extreme rankings (1 and 5) didn't allow a clear conclusion.

Lack of AngularJS experience. For 74.8 percent of the respondents, lack of AngularJS experience often caused performance problems (rankings of 4 and 5).

Inadequate application architecture. For 55.8 percent of the developers, an inadequate application architecture caused performance problems. An example of this problem involved an application not using caching when it had to compute a large list of items each time it presented the list to users.

AngularJS architecture. Regarding whether AngularJS's architecture caused poor performance, 40 percent of the respondents gave a neutral ranking (3), and no clear trend existed on either side of the scale.

The Problems' Technical Causes

The mapping study revealed 13 technical causes:

1. *Unnecessary two-way data binding*—for example, unnecessary synchronization between data in the view and in the model.
2. *The wrong watching strategy.* For example, during the digest cycle, values of the watched expressions might be compared by collection or value instead of by reference. (The digest cycle is the internal computation that automatically updates the view with changes detected in the model.) On one hand, this allows the comparison of nested values in objects. On the other hand, it requires a full traversal of the nested data structure on each digest. Moreover, a full copy of the data structure must be held in main memory.⁸
3. *Watching complex functions in the digest cycle.* If complex functions are registered for watching

- during the digest cycle, performance can drop considerably.
4. *Filters in templates* that are evaluated on each digest cycle.
 5. *Mouse event directives*, which can frequently trigger the digest cycle.
 6. *The wrong conditional-display strategy*. For example, the `ngIf` and `ngSwitch` directives create or destroy DOM (Document Object Model) elements, depending on a condition; these operations might be expensive.
 7. *Inappropriate use of `$scope.$apply()`*, which AngularJS uses to trigger the full digest cycle.
 8. *Runtime configurations and flags* that should be disabled but aren't when the application is in production.
 9. *Not cleaning up resources*, including handlers, watches, and asynchronous operations, even when they're no longer needed.
 10. *`ngRepeat` with long lists* rather than pagination or infinite scrolling.
 11. *`ngRepeat` without `track by`*. The `track by` expression avoids expensive operations—such as creation and deletion of DOM elements—by specifying a unique key for each item to make the identification.
 12. *Nested directives in `ngRepeat`*. This occurs when items of a collection used in an `ngRepeat` are represented by templates that create additional bindings. For instance, nesting four directives in an `ngRepeat` that's iterated 100 times would generate 400 new expressions that must be watched during the digest cycle.
 13. *Unconscious triggering of the full digest cycle*. Some methods that trigger the full digest cycle

are `$http`, `$resource`, and `$timeout`.

Not knowing this can lead developers to architect applications in which the digest cycle is triggered multiple times.

Most causes were related to the digest cycle. AngularJS provides the ability to constantly synchronize the application state with the view presented to the user. It does this by constantly observing the variables in the model. So, the more data to sync or the more complex the watched functions are, the slower this cycle is.

A closer look. We asked the developers to rate how often these items decreased performance, on a scale of 1 (very rarely) to 5 (very often). Figure 5 shows the results. The percentage of respondents who answered ranged from 70.5 for cause 8 to 87.4 for causes 1 and 3.

We also performed statistical tests on the data in Figure 5. Using the Shapiro-Wilk test, we found that the answers for every cause deviated from normality (the p -value ranged from $2.888e^{-5}$ to $7.162e^{-8}$). So, we used a Wilcoxon signed-rank test with a 95 percent confidence interval and the null hypothesis that the median of the answers was less or equal to 3. We rejected this null hypothesis for the first five causes in Figure 5 (1, 10, 11, 13, and 3). Therefore, we have statistical evidence that the respondents often or very often associated these causes with performance problems.

Moreover, we reran the test with the null hypothesis that the median of the answers was equal to or greater than 3. We rejected this null hypothesis for the last two causes in Figure 5 (8 and 5). We can state, with statistical support, that the

respondents rarely or very rarely associated these causes with performance concerns.

We first discuss the five causes that the respondents often or very often associated with performance problems.

The results for cause 1 (unnecessary two-way data binding) suggest that the ease of synchronizing the data between the view and the model—developers don't need to write any complex code—makes developers likely to overuse such a feature.

The results for cause 10 (`ngRepeat` with long lists) show that the respondents agreed that performance deteriorates when they render long collections with `ngRepeat`. This topic was consistently addressed in the documents in our mapping study and is one of the problems the AngularJS community tries to solve through third-party components. Moreover, developers can easily use `ngRepeat` to create complex DOM elements for each item in a collection, creating at the same time several bindings (watches) that slow down the digest cycle. A common solution to this problem includes reducing the displayed list through pagination or infinite scrolling.^{9,10}

The results for cause 11 (`ngRepeat` without `track by`) reveal how much attention developers must pay to the architecture of AngularJS applications when they're working with large amounts of data. Specifically, `track by` avoids expensive operations, such as creating and deleting DOM elements. It does this by defining a global identifier that allows the direct association of items in the server with the elements rendered in the client.

Cause 13 (unconscious triggering of the full digest cycle) relates to specific methods that intrinsically

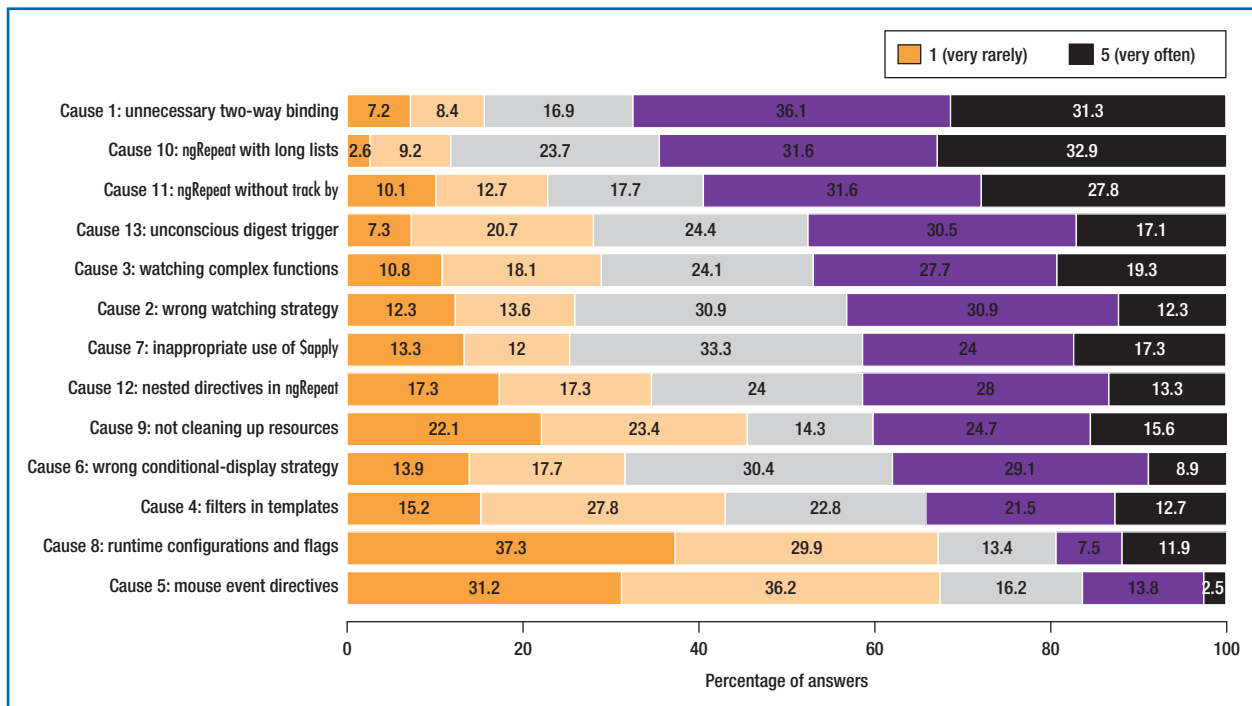


FIGURE 5. The occurrence of the 13 technical causes of performance problems in AngularJS. The respondents ranked the occurrence on a scale of 1 (very rarely) to 5 (very often). The causes are ordered from highest to lowest according to the percentage of answers with rankings of 4 and 5.

trigger the digest cycle, such as `$http`, `$resource`, `$timeout`, `$interval`, `$scope`, and `$apply`. The survey results suggest that it isn't trivial to track all the possible execution scenarios that involve these methods.

Cause 3 (watching complex functions) happens when developers register, through the `$watch` method, a function whose returned value is watched. In each digest cycle, the return values of the last and current executions are compared. If the function is complex, its execution will take longer, as will the digest cycle.

Now, let's examine the two causes that the respondents rarely or very rarely associated with performance problems.

Regarding cause 8 (wrong runtime configurations), the respondents didn't believe that disabling

debugging information boosted performance. In fact, developers would notice the effect of cause 8 only once, during application startup. This cause has little or no effect in the digest cycle, which might be why the respondents considered that disabling debugging information improved performance only slightly. In the mapping study, developers also reported that they perceived no improvements after disabling debug data.¹¹

The respondents might not have considered cause 5 (mouse event directives) serious because the problem appears only when these directives are spread throughout the application, creating multiple potential triggers of the entire digest cycle.

Overlapping results. To measure the strength of association between the

respondents' answers for pairs of causes, we ran Spearman's Rho test. For each pair of causes (C_i , C_j), we computed $\rho(A_i, A_j)$, where A_i and A_j are vectors with all the answers (on a scale of 1 to 5) that the participants gave to causes C_i and C_j . Only a moderate correlation ($\rho = 0.557$) existed between causes 2 (the wrong watching strategy) and 3 (watching complex functions). We expected this finding because both causes involve watching.

Threats to Validity

The web contains a vast amount of material on AngularJS, so our mapping study might have missed important documents. Also, when someone constructs a survey, there's always the risk of having ambiguous or unclear questions. We did our



MIGUEL RAMOS is a master's student in the Computer Science Department at the Federal University of Minas Gerais. His research interests include web development and software architectures. Contact him at miguel@dcc.ufmg.br.




MARCO TULIO VALENTE is an assistant professor and heads the Applied Software Engineering Research Group in the Computer Science Department at the Federal University of Minas Gerais. His research interests include software architecture and modularity, software maintenance and evolution, and software quality analysis. Valente received a PhD in computer science from the Federal University of Minas Gerais. He is a Researcher I-D of the Brazilian National Research Council and holds a Researcher from Minas Gerais State scholarship, from FAPEMIG. Contact him at mtov@dcc.ufmg.br; www.dcc.ufmg.br/~mtov.



RICARDO TERRA is an assistant professor in the Department of Computer Science at the Federal University of Lavras. His research interests include software architecture maintainability and evolvability. Terra received a PhD in computer science from the Federal University of Minas Gerais. Contact him at terra@dcc.ufla.br; www.dcc.ufla.br/~terra.

A new version of AngularJS (called Angular 2.0) was released in 2016. However, at least for a while, the two versions will be maintained in parallel because Angular 2.0 isn't backward-compatible with AngularJS 1.x. The most important architectural modification in Angular 2.0 is the new change detection mechanism, which uses immutable and observable data structures to detect changes to the model. Specifically, immutable objects reduce the number of checks when complex, nested data structures are compared for equality. Observables also help to improve performance by providing specific events to which other objects can subscribe in order to detect changes.

In future research, experiments can be conducted with real and benchmark applications to measure the performance gains achieved with immutables and observables. These experiments can also be extended to include other benchmarks designed to validate the developers' perceptions we reported in this article. 

Acknowledgments

FAPEMIG and the Brazilian National Research Council support our research.

References

1. "AngularJS API Docs," 2016; docs.angularjs.org.
2. M. Ramos et al., "AngularJS in the Wild: A Survey with 460 Developers," *Proc. 7th Workshop Evaluation and Usability of Programming Languages and Tools (PLATEAU 16)*, 2016, pp. 9–16.
3. C. Wohlin et al., *Experimentation in Software Engineering*, Springer, 2012.
4. K. Mustafa, "Angular Performance in ng-repeat," 2014; stackoverflow.com/q/22937504/5244036.

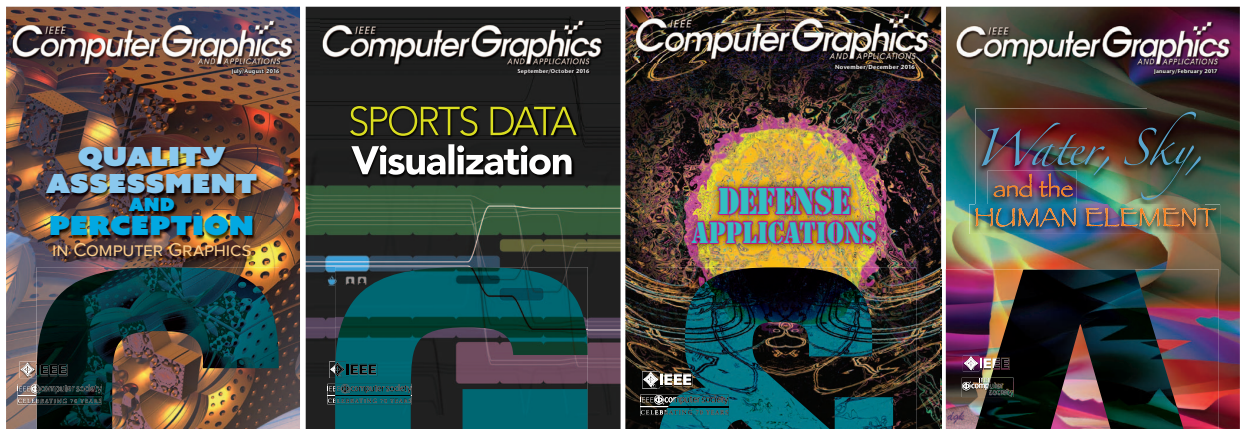
best to avoid this issue by using multiple iterations of small reviews when formulating our questions. Finally, the respondents might not be representative of the general population of AngularJS developers (external validity). To mitigate this threat, we advertised the survey in different online communities.

Our main findings are as follows. As we mentioned before, many developers reported that they inspected AngularJS code to fix or understand

performance issues. The most frequent general causes of poor performance were lack of AngularJS experience and inadequate application architectures. The most frequent causes of performance degradation in AngularJS applications were unnecessary two-way data binding and incorrect use of `ngRepeat`. As we also mentioned, the respondents didn't consider wrong runtime configurations and mouse event directives as major threats to performance. To explore and reinforce these findings, we plan to interview AngularJS developers.

5. “How to Improve Performance of ngRepeat over a Huge Dataset (Angular.js)?,” 2013; stackoverflow.com/q/17348058/5244036.
6. E. Koshelko, “Why You Should Not Use AngularJS,” 2015; medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99.
7. P. Koch, “The Problem with Angular,” 2015; www.quirksmode.org/blog/archives/2015/01/the_problem_wit.html.
8. “Scopes,” 2015; docs.angularjs.org/guide/scope.
9. C. Arora, “AngularJS Performance,” 2015; www.packtpub.com/books/content/angularjs-performance.
10. S. Fröstl, “AngularJS Performance Tuning for Long Lists,” blog, 2013; tech.small-improvements.com/2013/09/10/angularjs-performance-with-large-lists.
11. “AngularJS Disabling Debug Data in Production,” 2015; stackoverflow.com/q/32967438.

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>



CG&A
www.computer.org/cga

I IEEE *Computer Graphics and Applications* bridges the theory and practice of computer graphics. Subscribe to *CG&A* and

- stay current on the latest tools and applications and gain invaluable practical and research knowledge,
- discover cutting-edge applications and learn more about the latest techniques, and
- benefit from *CG&A*'s active and connected editorial board.