A Second Look at the Dynamics of the JavaScript Package Ecosystem

Kevin de Haan, Gregory Neagu, Frederic Sauve-Hoover, Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
Email: {kdehaan,neagu,rsauveho,abram.hindle}@ualberta.ca

***Abstract*—In recent years, the tools and packages most commonly involved with JavaScript development have evolved rapidly. Newer packages such as Angular and React have experienced a marked increase in popularity among developers, while frameworks such as jQuery have begun to phase out.[1] For this reason we take a second look at a 2016 paper by Wittern, Suter and Rajagopalan [14] to see what aspects of the JavaScript package ecosystem have changed, and if previously observed trends have remained constant. In the original paper the authors use the *node package manager* (`npm`) to gain insight into the JavaScript ecosystem as a whole, and data from projects publicly hosted on `GitHub` to observe an alternative measure of popularity. We adhere to the same methods of analysis, and extend the data to capture more recent information up to April 1st 2019. Ultimately, this second look aims to discover if recent years have had any significant effects on ecosystem-wide trends, and provide developers with further insight into how packages are used and evolve.**

***Index Terms*—JavaScript; Node.js; node package manager; software ecosystem analysis**

I. Introduction

Software ecosystems are environments that form as projects develop in parallel, becoming interconnected as contexts and dependencies span companies and communities [6]. Research on these systems has increased rapidly in the recent past [12], investigating their characteristics and behaviour as they develop [7]. Understanding how software ecosystems evolve is important from both a software as well as a business standpoint [8], and is valuable for informing developers how technologies are used over time [13]. An understanding of software ecosystems can inform decisions on when to adopt frameworks and how long to support them, as well as provide insight into how changes to software propagate throughout the community [14]. Additionally, determining the characteristics of software ecosystems can help clarify why some frameworks flourish while others fail, and guide developers in the creation of new tools [13]. Furthermore, because software projects are overwhelmingly a collaborative effort, a complete understanding of a single project often requires knowledge of the ecosystem supporting it [1].

This paper is a replication of *A Look at the Dynamics of the JavaScript Package Ecosystem* [14] that performs extensive analysis of the *node package manager* (`npm`) ecosystem. `npm` provides a set of open source tools that allow developers to describe packages for `Node.js`, an asynchronous JavaScript runtime environment designed for network applications[2]. The services provided by `npm` include a command line interface for maintaining `package.json` files, the primary method to describe package metadata such as the name, description, version, and dependencies of a given package. `npm` also allows developers to publish their packages to a public registry, permitting anyone to download and use their software. Packages hosted on `npm` will often depend on other `npm` packages, forming an elaborate JavaScript package ecosystem. Since the publishing of the original paper, the usage and scale of `npm` has only grown, and now hosts more than three times as many packages (over 750,000 as of April 1st 2019) and handles over ten times as many weekly package downloads (now over ten billion per week). Additionally, the major frameworks used in JavaScript development have undergone a rapid transformation as packages such as Angular and React are adopted[1]. The core contributions we make are as follows:

- We replicate and verify the results found in the original paper for the window of October 1st 2010 to September 1st 2015.
- We extend the analysis to the time period of September 2nd 2015 to April 1st 2019, and evaluate whether patterns and trends noted in the original paper are still observable.
- We investigate whether the continued evolution of the JavaScript package ecosystem has affected the relationships between various measures of package popularity.
- We determine if the ongoing maturation of the `npm` ecosystem has resulted in tangible changes to version numbering or adoption practices.

II. Data Collection

The window of data analyzed within this paper is October 1st 2010 (as in the original paper) to April 1st 2019. We collected from three publicly available data sets. Two of these, the `npm` registry and the `GitHub` repository platform, are from the same source as in the original paper. To find

[1]https://insights.stackoverflow.com/survey/2016#technology-most-popular-technologies, https://insights.stackoverflow.com/survey/2017#technology-_-frameworks-libraries-and-other-technologies, https://insights.stackoverflow.com/survey/2018#technology-_-frameworks-libraries-and-tools

[2]https://nodejs.org/en/about/

repositories relying on npm, we used the Google BigQuery github_repos data set, updated weekly[3]. By using this set we are able to analyze GitHub data without being constrained to the currently available window provided by the GHTorrent project [4]. The final data set encompasses 797,940 packages and **#VALUE** applications.

A. Package Metadata

```
{
  "name": "Lorem Ipsum",
  "version": "0.9.3",
  "maintainers": [
      {"name": "Dolor Sit",
      "email": "dolorsit@amet.org"}
  ],
  "repository": {
      "type": "git",
      "url": "https://github.com/lor/em"
  },
  "main" : "loremipsum.js",
  "keywords": ["Web", "REST"],
  "dependencies": {
      "Adipiscing": "~1.7.0",
      "Elit": "5.1.x"
  },
  "devDependencies": {
      "Sed": "0.9.0",
      "Do": ">=1.3.5 <4.0.0"
  }
}
```

Listing 1: A mock npm package.json. Some fields omitted for brevity.

```
{
  "name": "Lorem Ipsum",
  "versions": {
    "1.0.4": {
      "dependencies": {
        "Adipiscing": "~1.7.0",
        "Elit": "5.1.x"
      },
      "devDependencies": {
        "Sed": "0.9.0",
        "Do": ">=1.3.5 <4.0.0"
      }
    },
    "1.0.5": {
      "dependencies": {
        "Adipiscing": "~1.7.0",
      },
      "devDependencies": {
        "Do": ">=1.3.5 <4.0.0"
      }
    },
    "time": {
      "1.0.4": "2017-09-25T06:39:20.596Z",
      "1.0.5": "2018-05-22T08:35:40.227Z",
      "created": "2015-05-18T03:52:55.192Z"
    }
  }
}
```

Listing 2: A mock simplified npm metadata file.

Metadata for every package is available on the npm registry[4]. This metadata is a JSON file containing some info on licensing, documentation links, maintainers, and crucially for our purposes a log of each version of the package since it's creation, labelled using semantic versioning [11], and a time field containing the timestamp for each version. Each version in the version field contains various bits of version specific information such as authors, maintainers, license used, bugs, and a list of dependencies for that specific version.

We obtained the list of all npm package names from the npm registry skimdb[5], and then downloaded this metadata for all 797,940 packages on the npm registry, having found the names of all packages using the npm registry skimdb , which we then converted to a somewhat simplified form as in Listing 2 with just version and dependency information as this is all we care about for this study. There are two kinds of dependencies in the metadata files, a list of runtime dependencies and *devDependencies* used in testing and development.

We gathered some information on download counts for npm packages from the npmjs API[6] however we found that the npmjs API seems to no longer have any data on download counts before October 1 2017, and so were unable to collect historic download counts for npm as a whole or any specific packages.

B. Applications using npm Packages

To collect our data for package usage in public projects, we turned to the Google BigQuery github_repos data set[7]. From this data, we skimmed all projects and pulled those written in JavaScript, resulting in an initial total of 9,017,221 repositories. Simply being written in JavaScript is not enough to detect npm usage, so to find repositories using npm packages we looked for projects that contained a package.json file. After confirming that the formatting of the file matched the npm standard, we are able to confidently say that any remaining packages are using npm packages. With this criteria, we then compiled a list of [100,000 for now] project repositories from GitHub, a volume that we felt is likely to be an appropriate representation of all such public projects. Using the GitHub platform API and the list of valid repository names from before, we cloned the GitHub repository for each project. With this, we were then able to determine the date and time of every commit of the project, as well as the list of included npm packages at the time of commit, as listed in every historic version of package.json, providing us with the npm packages used by that repository at any given time in the project's history. The results showed that on average a repository had 37.9 commits as well as [STATS HERE]. Ultimately, of the 9,017,221 public

[3]https://github.com/fhoffa/analyzing_github/
[4]https://registry.npmjs.org/packageName
[5]https://skimdb.npmjs.com/registry/_all_docs
[6]https://api.npmjs.org/downloads/range/2010-01-01:2019-04-01/packageName
[7]https://bigquery.cloud.google.com/dataset/bigquery-public-dat:github_repos

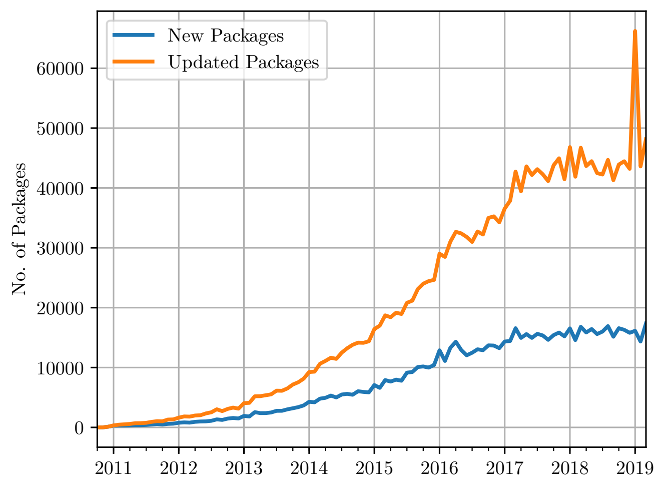

Fig. 1: Packages updated and packages created per month. Multiple updates per month are only counted once.

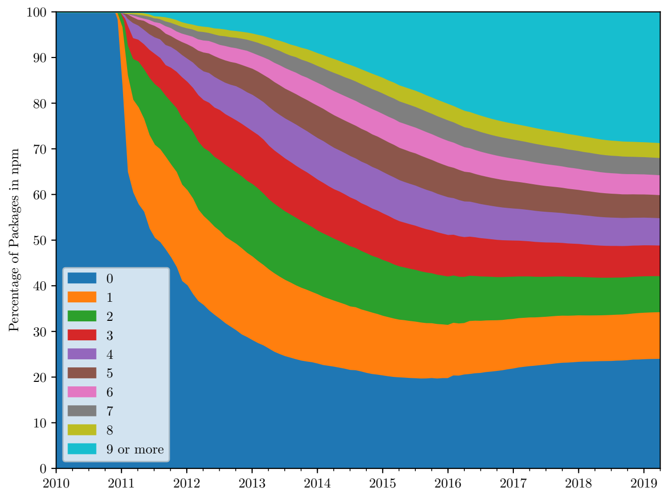

Fig. 2: npm packages by their number of dependencies on other packages

GitHub projects written in JavaScript and the [NUMBER] of repositories using npm packages, we randomly selected [NUMBER] repositories as a representative data set for the analysis.

III. Ecosystem Evolution

Created in 2009, npm has grown rapidly in popularity and scope over the last ten years, and as of the original paper showed no signs of slowing down [14]. We investigate the state of the npm ecosystem since September 1st 2015, and look for any signs of deterioration in the health of the ecosystem. Periods of stagnating growth would suggest that developer interest is waning, while steady activity would indicate that the ecosystem as a whole is healthy and will continue to evolve. To search for these potential indicators in the npm package environment, we investigate the number of packages created and updated over time, as well as system-wide trends of dependencies within packages. In Figure 1, we can observe that while npm is still growing, the speed at which new packages are created has slowed from an exponential to a linear rate. Some other things of note include an observable spike in package creation and updates around March 2016, the month when developer Azer Koçulu removed his 273 packages (including the popular package left-pad) from npm and in doing so affected some packages such as babel and atom, and therefore a significant fraction of *all* Node.js projects[8]. There is also a significant spike in package updates observed in the month of January 2019, caused by some source currently unknown to the authors [TRY TO FIGURE THIS ONE OUT]. [ADD CONCAVITY DATA]. Figure 2 [CURRENTLY MISSING][MORE DATA COMMENTARY].

To better visualize the status of inter-package dependencies, we constructed a directed dependency graph using dependants as out-degrees and dependencies as in-degrees. Based on this graph, Figure 2 displays the percentage of packages with various amounts of dependencies. [MIGHT NEED TO CHANGE BASED ON DATA: While the average number of dependencies of packages continues to increase, the percentage of packages with zero to three dependencies has actually increased since the end of the original paper's reporting period [14]. This suggests some changes to developer behaviour, likely either as a deliberate effort by programmers seeking to avoid the perils of complicated dependency trees [5], or as a natural result of the ever-changing ways in which JavaScript is used for project development. [ADD EXACT VALUES]

In addition to the number of external dependencies per package, we investigate how packages support dependants. The original paper discovered that the majority of in degrees are concentrated among a core minority of packages, with the majority of packages having no dependants, a discovery that has also been observed in other software ecosystems such as Linux and MySQL [9]. In our updated data, we discover that not only have these aspects held true, but the concentration of dependencies has actually increased. As of April 1st 2019, [NUMBER]% of packages had zero dependants, up from [NUMBER]% in 2015. Meanwhile, only %[NUMBER] of packages had 6 or more dependants, down from %[NUMBER] in September 2015. It is again interesting to note that the current trend of increasing concentration appears to have begun shortly after the end of window in the original paper [14].

IV. Package Popularity

As in the original paper we analyzed popularity of npm packages using three major measures

1) The **npm rank** is calculated using the PageRank [2] of an npm package within a dependency graph that we built using the package metadata as described in Section **??**. PageRank is calculated for every package using a

[8]https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm

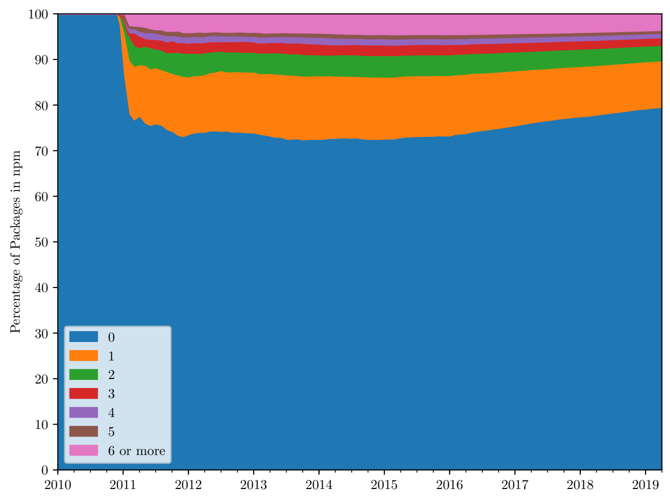

Fig. 3: npm packages by the number of other packages depending on them

Year	Top 10	Top 100	Top 250
2010	13	103	253
2011	25	297	701
2012	6	46	136
2013	3	38	114
2014	1	40	96
2015	4	38	86
2016	4	29	62
2017	0	12	48
2018	0	15	31
2019	0	4	12

Fig. 4: Number of packages entering top npm ranks for the first time

damping factor of 0.85 and stopping iterations once the cumulative change in values of vertices went below 10^{-6}. We then order all packages by their PageRank to determine the npm rank, an relative ranking of packge popularity with 1 being the most popular.

2) **download rank**

A. *Relationships between Measures*

[NEEDS MORE DATA]

B. *Distinct Package Types*

[NEEDS MORE DATA]

C. *Popularity Over Time*

[NEEDS MORE DATA]

1) Identifying Top Packages: [NEEDS MORE DATA]

2) Popular Package Dynamics: [NEEDS MORE DATA]

3) Comparing Popularities: [NEEDS MORE DATA]

V. Version Numbering and Package Adoption

[NEEDS MORE DATA]

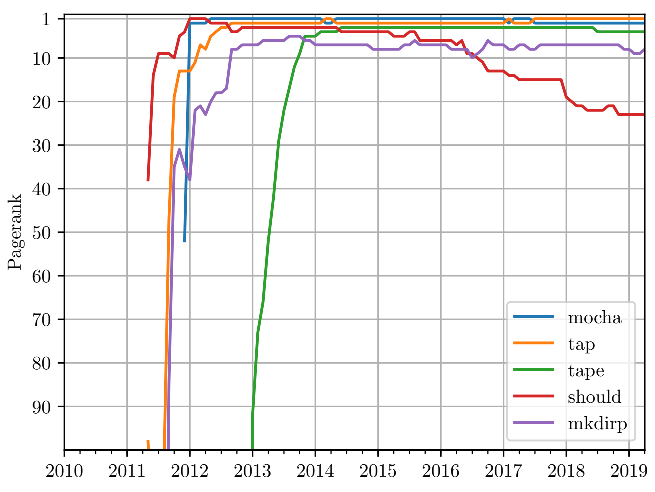

Fig. 5: npm rank (PageRank) of the five packages with the lowest overall geometric mean

A. *Attribution of Version Numbers*

[NEEDS MORE DATA]

B. *Adoption by Version Number*

[NEEDS MORE DATA]

VI. Related Work

[IN PROGRESS]

VII. Conclusion

[NEEDS MORE DATA]

References

[1] Kelly Blincoe, Francis Harrison, and Daniela Damian. Ecosystems in github and a method for ecosystem identification using reference coupling. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 202–207, Piscataway, NJ, USA, 2015. IEEE Press.

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[3] Alexandre Decan, Tom Mens, and Philippe Grosjean. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *CoRR*, abs/1710.04936, 2017.

[4] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[5] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 102–112, Piscataway, NJ, USA, 2017. IEEE Press.

[6] Mircea Lungu, Michele Lanza, Tudor Grba, and Romain Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4):264 – 275, 2010. Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008).

[7] Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems - a systematic literature review. *J. Syst. Softw.*, 86(5):1294–1306, May 2013.

[8] David Messerschmitt and Clemens Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. 01 2003.

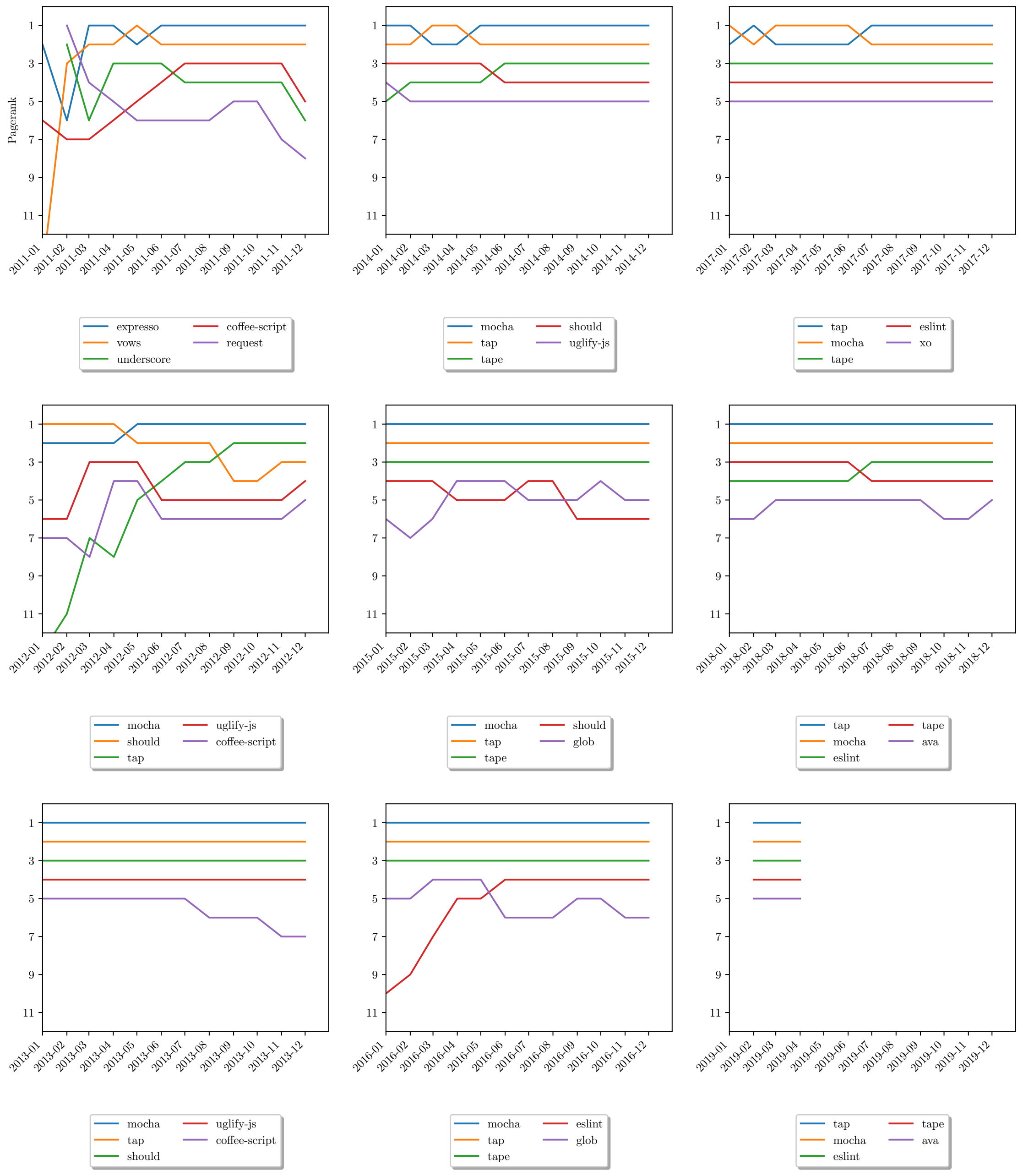

Fig. 6: `npm` rank (PageRank) of the five packages each year with the lowest geometric

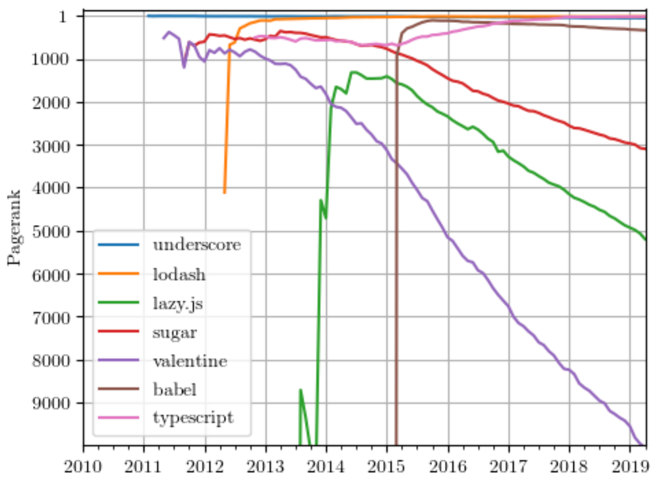

Fig. 7: `npm` rank (PageRank) over time of selected utility packages

[9] Christopher R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Phys. Rev. E*, 68:046116, Oct 2003.

[10] Amantia Pano, Daniel Graziotin, and Pekka Abrahamsson. What leads developers towards the choice of a javascript framework? *CoRR*, abs/1605.04303, 2016.

[11] Tom Preston-Werner. Semantic versioning 2.0.0.

[12] M. Seppnen, S. Hyrynsalmi, K. Manikas, and A. Suominen. Yet another ecosystem literature review: 10+1 research communities. In *2017 IEEE European Technology and Engineering Management Summit (E-TEMS)*, pages 1–8, Oct 2017.

[13] Alexander Serebrenik and Tom Mens. Challenges in software ecosystems research. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 40:1–40:6, New York, NY, USA, 2015. ACM.

[14] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. A look at the dynamics of the javascript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 351–361, New York, NY, USA, 2016. ACM.