A Second Look at the Dynamics of the JavaScript Package Ecosystem

Kevin de Haan, Frederic Sauve-Hoover, Gregory Neagu, Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
Email: {kdehaan,rsauveho,neagu,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 [23] 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 [12]. Research on these systems has increased rapidly in the recent past [21], investigating their characteristics and behaviour as they develop [13]. Understanding how software ecosystems evolve is important from both a software as well as a business standpoint [14], and is valuable for informing developers how technologies are used over time [22]. 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 [23]. 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 [22]. Furthermore, because software projects are overwhelmingly a collaborative effort, a complete understanding of a single project often requires knowledge of the ecosystem supporting it [3].

This paper is a replication of *A Look at the Dynamics of the JavaScript Package Ecosystem* [23] 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.

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 repositories relying on `npm`, we used the Google BigQuery `github_repos` data set, updated weekly[3]. By using this set

[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/

[3]https://github.com/fhoffa/analyzing_github/

we are able to analyze GitHub data without being constrained to the currently available window provided by the GHTorrent project [8]. The final data set encompasses 797,940 packages and 40,000 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 info on licensing, documentation links, and maintainers, as well as a log of each version of the package since its creation, which for our purposes is the most crucial element. The version data is labelled using semantic versioning [19] and includes a time field containing the timestamp of creation for each version. Each version in the version field contains various version specific information such as authors, maintainers, the 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]. Using these names, we then downloaded the metadata for each of the 797,940 packages hosted by the npm registry and then converted this metadata into a somewhat simplified form with just version and dependency information, as seen in Listing 2. There are two kinds of dependencies in the metadata files: runtime dependencies and devDependencies used in testing and development.

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[6]. 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. From this criteria, we compiled a list of applicable project repositories from GitHub. Using the GitHub platform API and the list of valid repository names from before, we cloned the GitHub repository for a random subset of all projects. With this, we were then able to determine the date and time of every commit of these projects, as well as the list of included npm packages at the time of every 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 to its package.json, and had 8.85 npm dependencies as well as 13.6 devDependencies. Ultimately, of the 9,017,221 public GitHub projects written in JavaScript and out of all of the repositories using npm packages, we selected 40,000 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 [23]. 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

[4]https://registry.npmjs.org/packageName

[5]https://skimdb.npmjs.com/registry/_all_docs

[6]https://bigquery.cloud.google.com/dataset/bigquery-public-dat:github_repos

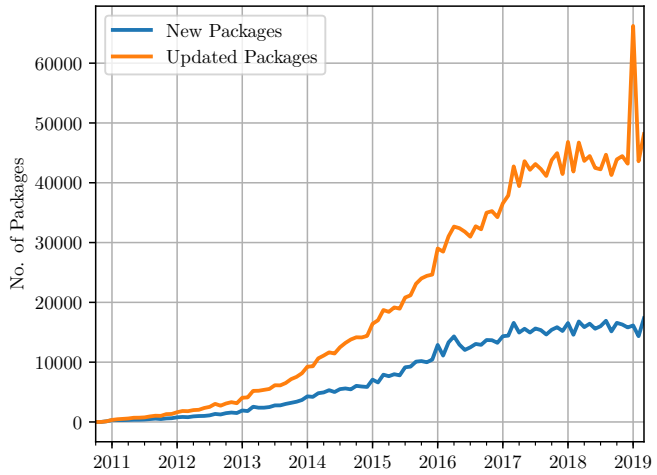

Fig. 1: Packages updated and packages created per month. Multiple updates per month are only counted once.

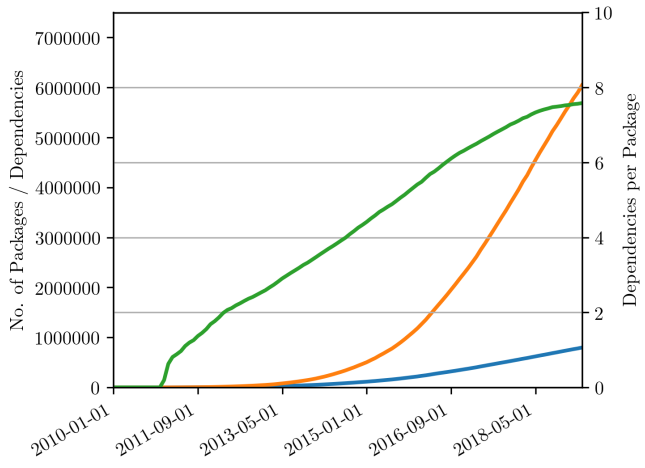

Fig. 2: Axis `y1` shows growth in number of `npm` packages (blue line), and the number of package dependencies (orange line) over time. Axis `y2` shows the average number of dependencies per package (green line).

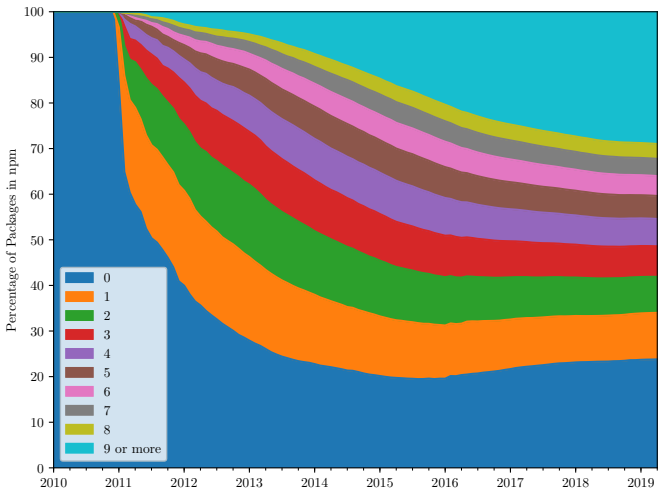

Fig. 3: `npm` packages by their number of dependencies on other packages.

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. Packages added and updated per month have gone from 10,087 and 23,100 respectively in September 2015 to 17,354 and 48,140 respectively as of April 2019. 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[7]. 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.

Figure 2 presents the total number of `npm` packages over time, as well as the total number of package dependencies. Additionally, we have plotted the average number of dependencies per `npm` package. Curiously, though total dependencies continues to grow faster than the creation of new packages, the number of dependencies for the average package appears to be plateauing. If this trend continues, it could suggest that the `npm` ecosystem is reaching some sort of equilibrium with regards to the number of package dependencies.

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 data, Figure 3 displays the percentage of packages with various amounts of dependencies. This confirms some of our earlier suspicious: 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 [23]. This supports the hypothesis that developer behaviour is changing, likely either as a deliberate effort by programmers seeking to avoid the perils of complicated dependency trees [10], or as a natural result of the ever-changing ways in which JavaScript is used for project development. In particular, the number of packages with zero dependencies has increased from 19.5% in September 2015 to 23.8% as of April 2019.

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

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

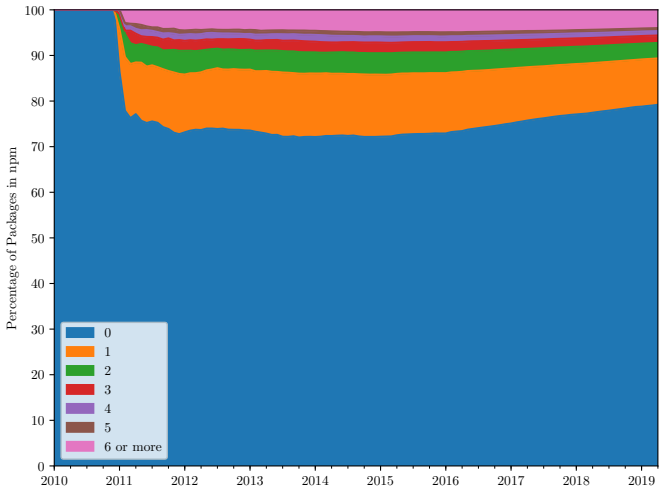

Fig. 4: npm packages by the number of other packages depending on them.

Linux and MySQL [16]. 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, 79.2% of packages had zero dependants, up from 72.8% in September 2015. Meanwhile, only 3.2% of packages had 6 or more dependants, down from 3.8% 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.

In section IV, we revisit the popularity measures investigated in the original paper [23] to discover changes in the relationships between package rankings both inside and outside of the npm ecosystem, gaining insight into the dynamics of npm package popularity.

Takeaways. We find that while the npm ecosystem continues to grow, the rate of growth is no longer superlinear, a possible sign of the ecosystem beginning to reach some form of maturity [1]. In the relationships between packages, we find that though the average number of dependencies continues to grow, there has been a recent decline in the rate of this growth. Accordingly, we have found that the the percentage of packages with zero dependencies has increased from 19.5% in September 2015 to 23.8% as of April 2019. The ratio of packages that are depended on has also declined, with the percentage of packages with one or more dependants decreasing from 27.2% in September 2015 to 20.8% as of April 2019. However, this still exhibits a power law relationship as observed in prior software ecosystem research [11], [16], [23].

IV. Package Popularity

As per the original paper we analyzed popularity of npm packages using three major measures [23]:

1) The **npm rank** is calculated using the PageRank [6] of an npm package within the dependency graph that we built using package metadata as described in Section II-A. PageRank is calculated for every package using a damping factor of 0.85, and stopping iterations once the cumulative change in values of vertices fell below 10^{-6}. We then order all packages by their PageRank to determine the npm rank, an relative ranking of package popularity with 1 being the most popular.
2) The **download rank** of a package is based on the the gross volume of package downloads from the npm registry, then ordered relatively with 1 being the most downloaded.
3) **GitHub rank** is a measure of the popularity of a package among public GitHub projects. This rank is created from a list of npm dependency occurrences found in `package.json` files, with rank 1 meaning that the package was the most common dependency among analyzed GitHub repositories.

	npm PageRank	Downloads	GitHub
npm PageRank	-	0.550	0.345
Downloads	0.550	-	0.444
GitHub	0.345	0.444	-

TABLE I: Spearman rank correlation coefficients between selected measures of popularity.

A. Relationships between Measures

In order to evaluate how different popularity metrics relate to each other and for what purposes they can be used, we investigate the relationships between our three chosen metrics. By calculating the Spearman rank correlation coefficients between the metrics, we are able to support the findings of the original paper for the subset of packages for which this correlation is applicable, i.e., only those packages which have assigned ranks for each measure used in calculating the correlation coefficient. The coefficients presented in Table I are similar to those found in the original paper, supporting the idea that each measure of popularity does not always represent the same kind of intrinsic properties of a package, and thus cannot necessarily be used interchangeably.

From these results, we can say that our data supports the findings of the original paper, and that our three metrics of popularity are measures of different package attributes. Functionally, this means that packages with a high rank in one metric can be ranked much lower by another metric.

B. Distinct Package Types

As we demonstrated in section IV-A, packages that perform well by one measure of popularity can perform poorly by others. The original paper proposes that one major source of this differential is in the way that packages are used [23]. It is surmised that packages ranked highly by the npm PageRank system are primarily packages that are used often within the npm ecosystem, and tend to be **core utility** packages that are used as dependencies for other npm packages, and rarely used outside of the npm package environment. Conversely, packages that exhibit significant popularity by the GitHub

Keyword	"npm strong"	"GitHub strong"	Diff.
	-	0.550	0.345
	0.550	-	0.444
	0.345	0.444	-

TABLE II: "npm strong" keywords ordered by difference in occurrence between correlated measures.

Keyword	"GitHub strong"	"npm strong"	Diff.
	-	0.550	0.345
	0.550	-	0.444
	0.345	0.444	-

TABLE III: "GitHub strong" keywords ordered by difference in occurrence between correlated measures.

rank tend to be **end user** packages more commonly used in application development as opposed to package construction. In order to confirm this hypothesis, we examine keywords associated with each ranking system, as is done in the original paper.

Tables II and III display keywords most associated with each popularity ranking. As the keywords in Table II are closely linked to low level utilities, and the keywords of Table III are strongly related to application frameworks, we feel confident confirming the hypothesis of the original authors.

C. Popularity Over Time

With the differences and changes in the relationships between popularity measures explored, we can now focus on npm rankings in particular, as it is most representative of the importance of a package to the npm ecosystem as a whole. To obtain npm ranks over time, we first begin with the complete dependency graph as of April 2019. We then examine the date references on the edges of the graph to produce PageRanks for each and every package on a monthly basis for the period of October 2010 to April 2019. From the list of PageRanks in each month, we then produce the relative rank for every package present in the npm ecosystem at that time.

1) Identifying Top Packages: One of the most obvious things to investigate is which packages have historically performed best throughout the entirety of npm's existence. To determine overall package popularity, we use the geometric mean npm rank of a package over the full window of analysis, and order them from lowest to highest (i.e, highly ranked to lowly ranked). Geometric mean is used in this instance instead of arithmetic mean in order to limit the effect of outliers [23]. With popularity calculated in this manner, Figure 5 presents the top five npm packages of all time.

The first thing to notice about these five packages is that the rankings have changed since the time of the original paper. Gone are the packages uglify-js and coffee-script, and newcomers tape and mkdirp have appeared. Additionally, though the package should remains on the list, it has fallen from the #2 to the #4 position.

Four of the five current champions are testing utilities—mkdirp is the sole exception, instead providing the ability to dynamically create directories from within node.js. Curiously, the testing packages are essentially all direct competitors

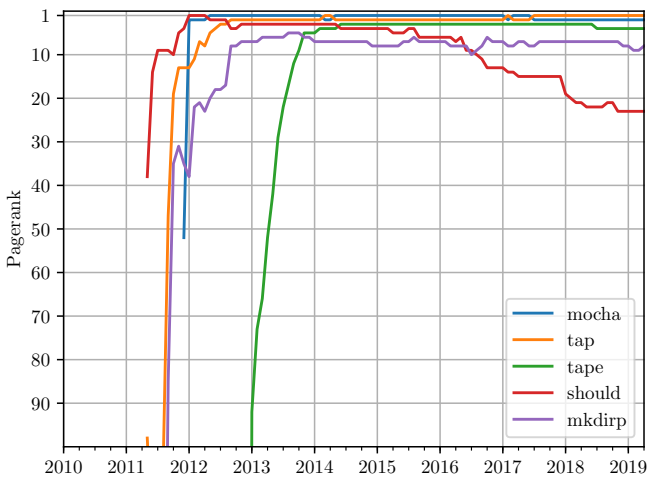

Fig. 5: npm rank (PageRank) of the five packages with the lowest overall geometric mean.

of one another, providing a powerful demonstration of both the widespread importance of testing utilities as well as the ability for the npm software ecosystem to support multiple core packages with similar yet distinct functionality.

While these packages represent the most consistently popular members of npm, we also evaluate the most popular packages on a yearly basis. Figure 6 presents the top five packages for each individual year from 2011 to 2019. As noted in the original paper, the top packages per year are relatively stable from 2013 onward [23]. One notable exception to this rule is the 2016 rise in popularity of eslint, a tool for improving the quality of JavaScript code[8].

2) Popular Package Dynamics: To better understand the evolution of package popularity, we investigate the dynamics of the top rankings over time. While Figure 6 demonstrates that package popularity appears to be relatively stable, we can examine the number of packages entering (and therefore exiting) the top n packages per year in order to obtain a better view of the ecosystem as a whole. Table IV depicts the amount of newcomers to the top npm ranks each year, further supporting the observation that core npm package popularity is becoming more stable as the ecosystem matures.

3) Comparing the Popularity of Similar Packages: One of the significant benefits of being able to determine npm ranks over time is that it allows for the comparison of similar packages against each other at an objective level. Figure 7 shows the npm ranks of selected utility packages, as chosen in the original paper [23]. One of these packages, underscore, is among the oldest npm utility packages, and the rest have been selected for being typical alternatives to the underscore package. Some of these packages, such as lodash, are directly or indirectly results of forks from the underscore repository. While lodash has kept up and even surpassed underscore in popularity, the majority of newer alternatives

[8]https://www.npmjs.com/package/eslint

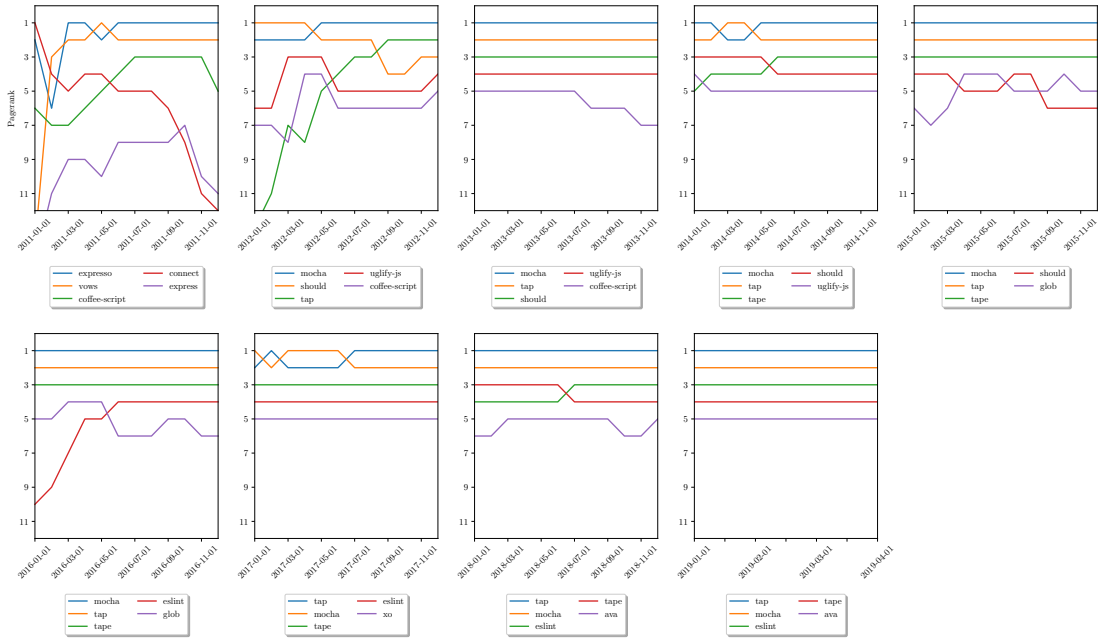

Fig. 6: `npm` rank (PageRank) of the five packages each year with the lowest geometric mean.

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

TABLE IV: Number of packages entering top `npm` ranks for the first time.

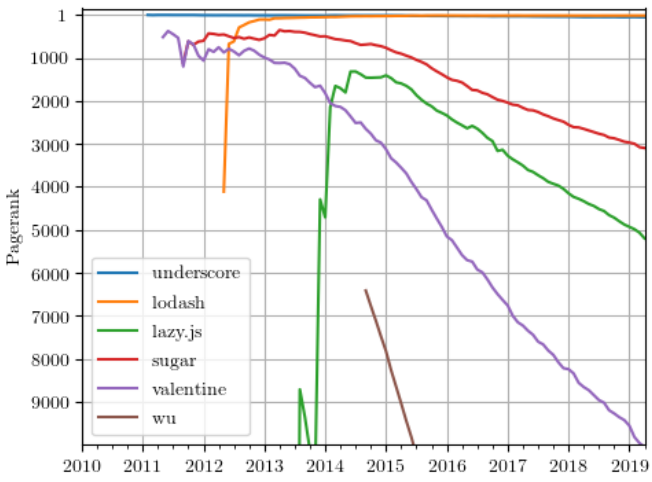

Fig. 7: `npm` rank (PageRank) over time of selected utility packages

are unable to consistently compete. Calculating ranks for similar groups of packages such as this allow us to isolate notable, successful packages across the entire span of the ecosystem's history, as well as evaluate the persistence of the most popular core packages over time. In the example group of packages presented in Figure 7, we can observe that the package established early in the ecosystem's life-cycle is well-positioned to maintain a significant level of popularity, while the majority of other packages tend to eventually decline unless they provide something exceptional, such as `lodash`[9] providing API compatibility with `underscore`[10].

Takeaways. Using `npm` ranks, we are able to identify successful packages at any point in the life-cycle of the ecosystem. We can also use these ranks to obtain a measure of package popularity stability over time. In the `npm` ecosystem, the amount of change in the highest ranked packages has decreased significantly since the inception of the service, with many more recent utilities falling out of favour relative to the oldest and most historically successful equivalents. However, there are still some relatively newer packages such as `lodash`

[9]https://lodash.com/
[10]https://underscorejs.org/

that have managed to compete with the oldest core packages such as `underscore`.

V. Related Work

Related research can be generally described as one of two types of work: meta-analyses of the behaviour and importance of software ecosystems, and empirical investigations into the specifics of a particular ecosystem. Some topics for meta-analyses include ecosystem visualization [12], ecosystem maturity [1], quality metric aggregation [15], and literature reviews [13], [21]. Specific ecosystem analysis include works such as this paper as well as the original research by Wittern, Suter and Rajagopalan [23] on `npm`. Others have performed evaluations of specific ecosystems such as Raemaekers et al. [20], who present a dataset with basic metrics, dependencies and changes within the popular Java-based Maven ecosystem. Another work investigates the result of changing project interdependencies in Apache [2], and still others evaluate evolution over time in ecosystems like Gentoo [4], Ruby, [9], and R [18]. Some works have looked empirically at the results of changing versions within software ecosystems (including `npm`), finding that developers often encounter difficulties when attempting to avoid breaking dependencies [5].

Finally, `npm` has been investigated informally. In the `pagerank` package, web-based PageRanks are used to evaluate package popularity[11]. The project `npm-by-the-numbers` also provides various statistics about the state of `npm`, but is significantly out of date and in contrast to this work does not provide an evaluation of the ecosystem over time or give any insight into application usage[12].

VI. Conclusion

In this paper, we take a second look at the `npm` ecosystem, replicating and extending the results of previous research by Wittern, Suter and Rajagopalan [23]. Though we observe that the growth of `npm` has slowed down, the ecosystem continues to thrive, demonstrating signs of maturity rather than stagnation.

Contrary to the trend observed in the original paper, there now appears to be some effort to reduce the number of inter-package dependencies, and therefore the risks associated with complex dependency trees. The core set of packages upon which multiple other projects rely on has remained relatively constant in size, and demonstrates a trend towards even further concentration. Accordingly, the set of packages most depended on in the `npm` ecosystem is also trending towards increasing stability, with fewer and fewer packages entering the top ranks each year.

Our results regarding relationships between popularity metrics support the discoveries of the original paper, further reinforcing the theory that packages can be generally divided into the categories of end user and core utility packages, and improving our understanding of how package popularity evolves. Ultimately, our second evaluation of the `JavaScript` package environment can help developers evaluate the state of the `npm` ecosystem, and continue to make informed decisions when selecting frameworks to use in their applications.

[11] https://www.npmjs.com/package/pagerank

[12] https://github.com/ashleygwilliams/npm-by-the-numbers

References

[1] Salviano C.F Alves A.M., Pessoa M. *Towards a Systemic Maturity Model for Public Software Ecosystems*, volume 155. Springer, Berlin, Heidelberg, 2011.

[2] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. The evolution of project inter-dependencies in a software ecosystem: The case of apache. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 280–289, Washington, DC, USA, 2013. IEEE Computer Society.

[3] 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.

[4] Remco Bloemen, Chintan Amrit, Stefan Kuhlmann, and Gonzalo Ordonez-Matamoros. Gentoo package dependencies over time. pages 404–407, 05 2014.

[5] Christopher Bogart, Christian Kästner, and James Herbsleb. When it breaks, it breaks: How ecosystem developers reason about the stability of dependencies. In *Proceedings of the ASE Workshop on Software Support for Collaborative and Global Software Engineering (SCGSE)*, pages 86–89, 11 2015.

[6] 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.

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

[8] 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.

[9] Jaap Kabbedijk and Slinger Jansen. Steering insight: An exploration of the ruby software ecosystem. volume 80, pages 44–55, 06 2011.

[10] 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.

[11] Panagiotis Louridas, Diomidis Spinellis, and Vasileios Vlachos. Power laws in software. *ACM Trans. Softw. Eng. Methodol.*, 18(1):2:1–2:26, October 2008.

[12] 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).

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

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

[15] Karine Mordal-Manet, Nicolas Anquetil, Jannik Laval, Alexander Serebrenik, Bogdan Vasilescu, and Stphane Ducasse. Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process*, 25:1117–1135, 10 2013.

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

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

[18] Konstantinos Plakidas, Daniel Schall, and Uwe Zdun. Evolution of the r software ecosystem. *J. Syst. Softw.*, 132(C):119–146, October 2017.

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

[20] Steven Raemaekers, Arie van Deursen, and Joost Visser. The maven repository dataset of metrics, changes, and dependencies. pages 221–224, 05 2013.

[21] 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.

[22] 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.

[23] 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.