What makes a good Node.js package? Investigating Users, Contributors, and Runnability

BODIN CHINTHANET, Nara Institute of Science and Technology, Japan

BRITTANY REID, CHRISTOPH TREUDE, and MARKUS WAGNER, University of Adelaide, Australia

RAULA GAIKOVINA KULA, TAKASHI ISHIO, and KENICHI MATSUMOTO, Nara Institute of Science and Technology, Japan

The Node.js Package Manager (i.e., npm) archive repository serves as a critical part of the JavaScript community and helps support one of the largest developer ecosystems in the world. However, as a developer, selecting an appropriate npm package to use or contribute to can be difficult. To understand what features users and contributors consider important when searching for a good npm package, we conduct a survey asking Node.js developers to evaluate the importance of 30 features derived from existing work, including GitHub activity, software usability, and properties of the repository and documentation. We identify that both user and contributor perspectives share similar views on which features they use to assess package quality. We then extract the 30 features from 104,364 npm packages and analyse the correlations between them, including three software features that measure package "runnability"; ability to install, build, and execute a unit test. We identify which features are negatively correlated with runnability-related features and find that predicting the runnability of packages is viable. Our study lays the groundwork for future work on understanding how users and contributors select appropriate npm packages.

CCS Concepts: • **Software and its engineering** → **Software libraries and repositories**.

Additional Key Words and Phrases: Open source software, software libraries, software ecosystems

1 INTRODUCTION

The Node.js Package Manager (i.e., npm) serves as a critical part of the JavaScript community and provides support to one of the largest developer ecosystems in the world, with over 1.7 million packages [44]. These packages provide developers with useful features and libraries without the need to "reinvent the wheel", with each package often depending on several others. Searching for a suitable third-party library is a well-known problem for software developers [64], especially in a massive network like the npm ecosystem.

Authors' addresses: Bodin Chinthanet, bodin.chinthanet.ay1@is.naist.jp, Nara Institute of Science and Technology, Nara, Japan; Brittany Reid, brittany.reid@adelaide.edu.au; Christoph Treude, christoph.treude@adelaide.edu.au; Markus Wagner, markus.wagner@adelaide.edu.au, University of Adelaide, Adelaide, Australia; Raula Gaikovina Kula, raula-k@is.naist.jp; Takashi Ishio, ishio@is.naist.jp; Kenichi Matsumoto, matumoto@is.naist.jp, Nara Institute of Science and Technology, Nara, Japan.

Most research revolves around selection criteria based on quantifying a library goodness of fit for a certain scenario. For example, works such as LibRec [59] and other proposed library recommendation techniques present a scenario where a developer is interested in using a library that has also been used in other similar projects [13, 28, 42, 45, 47, 55]. Other related work focuses on the motivations behind why a software developer selects a package. For example, Larios Vargas et al. [36] argue that software developers often choose libraries arbitrarily, without considering the consequences of their decisions. These studies confirm that developers do struggle with library selection and updates [17, 28, 48, 63, 69]. In all of these works, the common assumption is that the *interested audience* of a library is a typical software developer who would like to adopt the package into their application.

Unlike related work, the novelty of this work is to provide a comprehensive investigation of features from previous work to quantify the goodness of a package by analysing the interested audience perspective. Based on the literature, we identified the audience from two overlapping perspectives:

(1) *Users* - who are looking to adopt a package into their applications. They are typically software developers [5, 36], who are interested in how well documented a package is and how it can be integrated into their existing project.

(2) *Contributors* - who are interested in contributing to a package. They are likely to be newcomer developers who view the package as a software project they would like to onboard [57, 58].

To characterise Node.js packages published on npm registry (i.e., npm packages) from these two perspectives, we conducted an online survey to ask Node.js developers what features they consider before adopting or contributing to assessing the quality of npm packages. We separate features of npm packages into four types: (1) *Documentation*: the properties of the package documentation, (2) *Repository*: the properties of the git repository, (3) *Software*: the ability to install, build, and execute the test of the package, and (4) *GitHub activity*: the interaction between developers and the repository on GitHub. We formulate the first research question as follows:

(RQ1) *Which features of an npm package do practitioners find relevant for assessing package quality?*

The key results of RQ1 are that (1) users and contributors of npm packages share similar views on which features are important for selecting a good package; users focus more on how to use the package, while contributors focus on the contribution guideline and how to build and test the package, (2) developers agree that software and documentation features are highly relevant to assess package quality, and (3) developers believe that repository features do not belong to any perspectives and do not reflect the package quality.

To investigate what trade-offs exist between features, we collected diverse data related to Node.js packages to create GH-Node.js, a dataset of 723,218 Node.js package repositories. First, we extracted and analysed the npm package features from 104,364 npm packages to understand the correlation among the features. We then explored the possibility of predicting package runnability since software features are highly relevant for assessing package quality, as shown in our survey. Through the two lenses of perspectives and extracted features, we formulate the two following research questions to guide our study:

(RQ2) *What features of an npm package correlate from different perspectives?*

The key results of RQ2 are that (1) features from the same type usually have strong correlations. (2) Software features are negatively correlated with other features. (3) Features that are less likely to be considered by any perspective, such as a number of files, are correlated with the ability to build a package.

(RQ3) *What features of an npm package predict whether it is runnable or not?*

The key results of RQ3 are that (1) predicting runnability of the package is viable. (2) Based on the permutation feature importance, we find that repository features are important for predicting the runnability.

In summary, this paper presents the following contributions:

- A definition of the 'goodness' of an npm package from two perspectives, user and contributor.
- A survey of Node.js developers to investigate how different perspectives select npm packages.
- Three measures of the runnability of an npm package.
- GH-Node.js, a dataset that contains a curated set of 723,218 Node.js packages containing repository and social interaction information.
- A large scale analysis of 104,364 npm packages for correlations and predictions of runnability.

2 AUDIENCE PERSPECTIVES OF AN NPM PACKAGE

In this paper, we characterise the audience of an npm package into two different perspectives. These perspectives are based on the various ways in which npm packages or software projects, in general, have been discussed in related work:

(1) *User perspective* - according to Lethbridge et al. [37], apart from very good code design and reusability aspects, high quality documentation is the key to learning a software system. Bennett [5] explained that "The greatest library in the world would fail if the only way to learn it was reading the code (and, in fact, it already has to a large extent). Some packages have managed to overcome this by way of lots of unofficial documentation – blog entries

Table 1. List of 30 features for package quality assessment presented in the survey. The features are grouped by their characteristic (feature type).

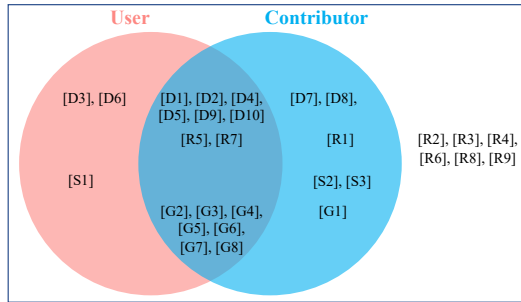| Feature type	Feature	Description
Documentation	[D1] hasAReadmemd	the existence of the README.md file in the root directory of the package [30, 51]
	[D2] linesInReadmemd	the number of lines in the README.md file [30, 51]
	[D3] numberOfCodeBlocks	the number of code snippets extracted from the README.md file [30, 51]
	[D4] readmemdCodeSnippetsNumber	the number of JavaScript snippets in the README.md file [30]
	[D5] hasAnInstallExample	the existence of an install example in the README.md file [24, 30, 51]
	[D6] hasARunExample	the existence of a run example in the README.md file [30, 51]
	[D7] hasAContributionGuideline	the existence of a CONTRIBUTING.md file in the root directory of the package [18, 34, 51, 56]
	[D8] hasACodeOfConduct	the existence of a CODE_OF_CONDUCT.md file in the root directory of the package [51, 61]
	[D9] hasALicense	the package has either license file or license description in REAMDE.md file [3, 30, 41, 51]
	[D10] githubLink	the GitHub repository link of the package [30]
Repository	[R1] hasASourceDirectory	the existence of a source directory (i.e., /src) within the root directory of the package [32]
	[R2] numberOfFiles	the number of files in the package [4]
	[R3] numberOfJsFiles	the number of JavaScript files (.js) in the package [4]
	[R4] numberOfHtmlFiles	the number of HTML files (.html) in the package [4]
	[R5] sizeOfRepository	the size of the package repository in bytes [43]
	[R6] mostPopularFileExtension	the most popular file extension appearing in the repository [4]
	[R7] hasATestDirectory	the existence of a test directory (i.e., /test, /tests) in the root directory of the package [41]
	[R8] numberOfRepositoryTags	the number of git tags in the repository [31]
	[R9] numberOfCommits	the number of commits in the main branch of the package [4]
Software	[S1] ableToInstall	whether the package is able to install the project [38]
	[S2] ableToBuild	whether the package is able to build (i.e., install dependencies, prepare working environment) [4, 21]
	[S3] ableToExecuteATest	whether the package is able to execute all test cases [4]
GitHub activity	[G1] numberOfNewcomerLabels	the number of newcomer labels in the GitHub issues [58]
	[G2] numberOfContributors	the number of GitHub contributors in the package [22, 41]
	[G3] numberOfIssues	the number of GitHub issues [22, 68]
	[G4] numberOfPullRequests	the number of GitHub pull requests [22]
	[G5] mostRecentIssue	the most recent GitHub issue date in an epoch format [22]
	[G6] mostRecentPullRequest	the most recent pull request date in an epoch format [22]
	[G7] latestRepositoryUpdate	the most recent committed date in an epoch format [22]
	[G8] latestReadmemdUpdate	the most recent README.md update date in an epoch format [22]

Fig. 1. Respondents mapping of features to the perspectives.

and the like – but there is absolutely no substitute for full, well-written documentation." Therefore, we consider the user perspective essential to our investigation of what makes a good Node.js package.

(2) *Contributor perspective* - according to Steinmacher et al. [57], newcomers can be attracted to a project with characteristics such as license type, project visibility, or a number of existing contributors. Moreover, providing good support for the onboarding of newcomers to contribute to a project is important. It can be done by identifying good first issues and creating informative descriptions for them [58]. A package that is good for users (e.g., easy to install) might not necessarily be good for contributors (e.g., presence of a CONTRIBUTING.md) and vice versa. Investigating such different perspectives is one of the goals of this work.

2.1 Developer Survey on Perspective

To answer (RQ1) *Which features of an npm package do practitioners find relevant for assessing package quality?* we start our investigation with a list of 30 package features derived from related work. We extracted these features from papers on npm in particular or software reuse in general. Table 1 shows the 30 features we identified along with the reference(s) for each. We group the features into four types based on their characteristics: (1) *Documentation*, (2) *Repository*, (3) *Software*, and (4) *GitHub activity*. However, related work has not validated the usefulness of these features to contributors and users. To address this, we conducted an online survey to identify developer opinions on what perspectives npm features belong to and what features are relevant for assessing npm package quality.

In the survey, we first asked the developers "which perspective do these features belong to?" for each of the 30 features, to which they can answer either (1) user perspective, (2) contributor perspective, (3) both of them, or (4) none of them. To summarise the result of this question, we assign the top voted choice as the perspective of each feature. In the case of the top voted choices are "user" and "contributor" (i.e., scores are tied), we assign "both of them" to the feature. We also confirmed that there were no cases where the "none of them" choice tied with others. After that, we asked the developers about their demographics, i.e., their experience with Node.js and npm packages, including their contributions on GitHub. Next, we asked developers to evaluate "how relevant are these features for assessing package quality?" for each of the 30 features using a five-point Likert scale, i.e., ranging from strongly agree to strongly disagree with a neutral option the middle strongly. To find participants for the survey, we contacted 2,150 npm developers and received 33 responses.

Table 2 shows the demographics of the participants. All have at least two years of experience with Node.js and npm. 19 developers described themselves as only a user of packages, two as a

Table 2. Demographic of the respondents (33 Node.js developers).

How many years of experience do you have with Node.js?	
2 to 3 years	3
4 to 5 years	11
6 to 7 years	12
8 to 9 years	3
10 years or more	4
What do you identify yourself as?	
User	19
Contributor	2
Both	12
How often do you use npm packages in your projects?	
Once a month or more often	30
Less than once a month but more than once per year	2
Less than once per year	1
How often do you contribute to npm packages on GitHub?	
Once a month or more often	8
Less than once a month but more than once per year	18
Less than once per year	6
Never	1

contributor to packages and 12 as both a user and contributor. Regarding the usage of npm packages, 30 developers responded that they used packages once a month or more often, two used them less than once a month but more than once per year, and the only one used npm packages less than once per year. Eight developers contribute to packages on GitHub once a month or more often, 18 less than once a month but more than once per year, six contribute less than once per year, and only one never makes any contributions.

Figure 1 shows which perspective package features belong to according to the opinion of developers. We find that half of the features are shared among both user and contributor perspectives. If features are specific to either perspective, users focus on the example of code or snippet (D3, D6) and ability to install the package (S1). On the other hand, contributors focus on guideline and code of conduct (D7, D8), source code directory (R1), ability to build and test (S2, S3), and newcomer labels in GitHub issues (G1). Interestingly, most repository features (R2, R3, R4, R6, R8, R9) do not belong to any perspectives since their usefulness does not convince participants to assess the package quality.

Figure 2 shows the survey results on how relevant the features are for assessing package quality. The green, grey, and yellow bars represent the level of agreement, neutrality, and disagreement from developer votes. Overall, we find that developers agree that documentation, GitHub activity, and software features could be used to measure the quality of npm packages – on average, 73%, 43.5%, and 79% respectively agree (i.e., agree and strongly agree). There are three features that get more than 90% of agreement includes (1) hasARunExample (100%), (2) hasAReadmemd (100%), and (3) ableToInstall (97%). On the other hand, developers disagree that repository features can reflect

(a) Documentation features (Median agreement: 73%, neutral: 24%, disagreement: 6%)

	Disagreement	Neutral	Agreement
hasARunExample	0%	0%	100%
hasAReadmemd	0%	0%	100%
hasALicenseFile	6%	9%	85%
hasAnInstallExample	6%	12%	82%
numberOfCodeBlocks	0%	27%	73%
githubLink	3%	24%	73%
readmemdCodeSnippetsNumber	6%	39%	55%
hasAContributionGuideline	18%	24%	58%
linesInReadmemd	15%	33%	52%
hasACodeOfConduct	27%	36%	36%

(b) Repository features (Median agreement: 27%, neutral: 34.5%, disagreement: 42%)

	Disagreement	Neutral	Agreement
hasATestDirectory	12%	6%	82%
hasASourceDirectory	27%	24%	48%
sizeOfRepository	36%	30%	33%
numberOfRepositoryTags	39%	30%	30%
numberOfCommits	42%	30%	27%
mostPopularFileExtension	42%	45%	12%
numberOfJsFiles	45%	42%	12%
numberOfFiles	45%	42%	12%
numberOfHtmlFiles	48%	39%	12%

(c) Software features (Median agreement: 79%, neutral: 15%, disagreement: 6%)

	Disagreement	Neutral	Agreement
ableToInstall	0%	3%	97%
ableToExecuteATest	6%	15%	79%
ableToBuild	6%	15%	79%

(d) GitHub activity features (Median agreement: 43.5%, neutral: 37.5%, disagreement: 19.5%)

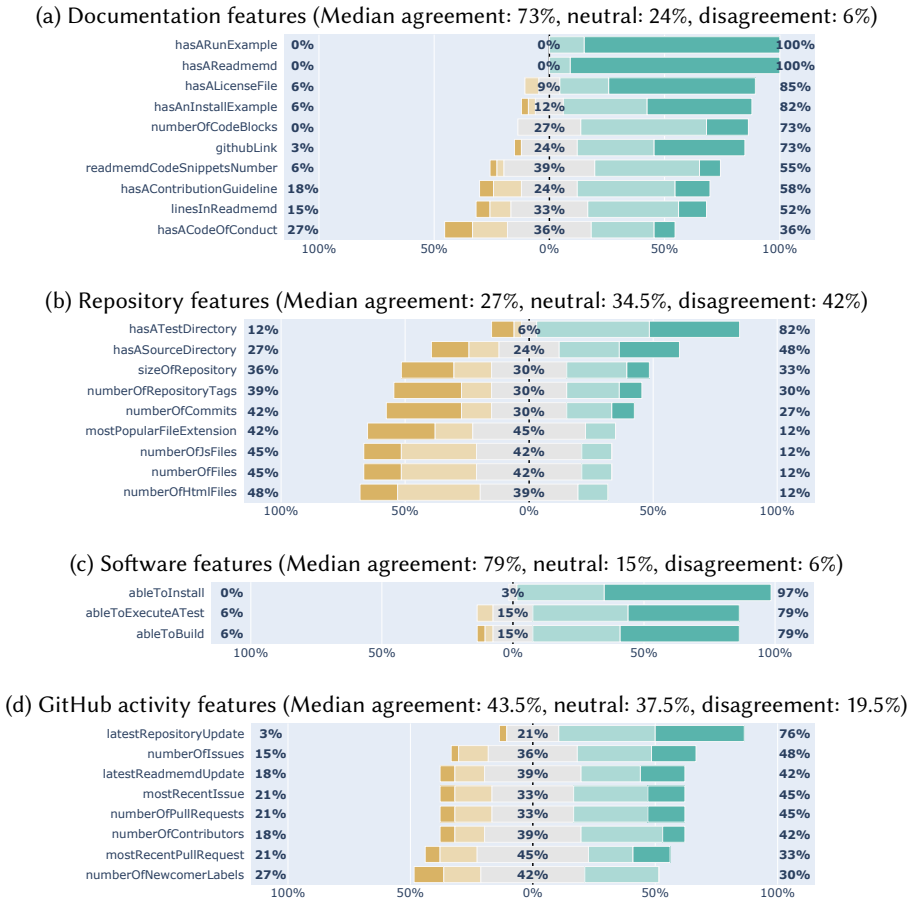|	Disagreement	Neutral	Agreement
latestRepositoryUpdate	3%	21%	76%
numberOfIssues	15%	36%	48%
latestReadmemdUpdate	18%	39%	42%
mostRecentIssue	21%	33%	45%
numberOfPullRequests	21%	33%	45%
numberOfContributors	18%	39%	42%
mostRecentPullRequest	21%	45%	33%
numberOfNewcomerLabels	27%	42%	30%

Fig. 2. Survey results on the relevance of features for package quality assessment. Left hand (yellow) shows levels of disagreement, middle (grey) shows neutral, and right (green) shows levels of agreement.

the quality of packages – on average, 42% of the respondents disagree (i.e., disagree and strongly disagree).

Summary: Assessing a package quality from both perspectives is slightly different. Users focus on how to use the package, while contributors focus on the contribution guidelines and how to build and test the package. Developers agree that software and documentation features are highly relevant for package quality assessment; in contrast, repository features are not.

3 GH-NODE.JS: A NODE.JS REPOSITORY AND INTERACTION DATASET

To extract the required features needed by the two perspectives, we compose our own dataset of npm repositories. GH-Node.js is an open dataset contains a curated snapshot of Node.js and npm packages that focuses on the social, technical, and documentation aspects. GH-Node.js is based on

Table 3. Dataset Snapshot Statistics. The full dataset estimations are approximate values.

Node.js and npm Package Repository Information	
Repository Snapshot	Nov 2, 2020
# Node.js packages	723,218
# total npm packages	104,364
- # last update 2013	7
- # last update 2014	9,867
- # last update 2015	25,680
- # last update 2016	26,115
- # last update 2017	15,057
- # last update 2018	8,869
- # last update 2019	7,642
- # last update 2020	11,127
- # commits	6,402,982
- # repository tags	674,258

two existing datasets (GHTorrent and Libraries.io). GHTorrent provides a mirror of git repositories and developer interactions gathered from GitHub [22], while Libraries.io provides meta-data and relationships among packages hosted on popular software ecosystems, e.g., npm, Maven, and PyPI [60].

Table 3 shows the summary statistics of GH-Node.js from November 30, 2020, with 723,218 Node.js repositories, which only 104,364 repositories are identified as npm packages. We also collected 1,960,345 issues, 1,083,828 pull requests, and 283,360 contributors associated with npm packages by using the GitHub API. In total, we took four months, from August to December 2020, to acquire and process all information for GH-Node.js. The dataset is available at this following link: https://doi.org/10.5281/zenodo.5010160.

4 EXPERIMENT SETUP

To answer RQ2 and RQ3, we extracted a subset of GH-Node.js as shown in Table 4. For our data preparation, we first extract metrics that relate to each perspective. Note that to answer RQ2, we focus on all 11,127 packages that were last updated in 2020 from the total of 104,364 packages. Our motivation was to retrieve projects that were likely to be active.

4.1 Runnable Code: Build and Run Tests

Building a package (79%) and running its unit tests (also 79%) are two features that received positive feedback in RQ1 survey. They are also among the first tasks that contributors do to ensure that the selected package is ready to run and test new features or fixes in the local environment. To confirm that an npm package is runnable from its source code, we automatically create a virtual environment, build, and test the package from scratch.

Our approach to building and testing each package from its repository consists of four steps:

(1) Create a docker container to construct an isolated environment.
(2) Clone the repository of the npm package to the docker container.
(3) Build the repository by using npm install --no-audit.
(4) If the package is built successfully, then execute npm test to run a unit test script as detailed in the package.json file.

Table 4. Dataset for our experiment.

Runnable Code	
# repositories (last updated in 2020)	11,127
# successfully built	8,199
- # passed tests	607
- # failed tests	4,262
- # no test	3,330
# unsuccessfully built	2,928
Executable Documentation	
# repositories	104,364
# collected code snippets	233,826
- Max	302
- Min	0
- Mean	2.2405
- Median	1.0000
- SD	4.8990
# successfully installed	97,006
# executed code snippets	220,324
- # successfully executed	33,484
- # unsuccessfully executed	186,840

From our trial and error, we set a timeout of five minutes for processing to detect frozen processes.

As shown in Table 4, we selected 11,127 repositories with at least one commit in 2020 as the input of the runnable code experiment. We find that 8,199 packages (73.68%) are successfully built. From these built packages, only 607 packages (7.40%) have at least one test case and are able to pass all their test cases. The rest of successful built packages are failed to pass any test cases (51.98%) or no test case available (40.61%). Note that the building and the testing process took around 21 days of execution.

4.2 Runnable Package: Install and Execute Code Snippets

Installing a package (97%) and trying the run example (100%) are two features from the user perspective, which also received a positive result from RQ1. Both features are among the first tasks that package users do to ensure that the selected package is executable in their applications. To confirm that an npm package is runnable after installed in any application, we automatically create a virtual environment, install, and execute the extracted code snippets.

We extracted 233,826 Node.js code snippets from the README files of all 104,364 repositories (last updated from 2013 up to 2020). Example code in markdown files can be identified by the surrounding special characters (```) that allow for the rendering of code blocks, and additional language information may be provided to allow for syntax highlighting; for example, adding the tag js to the same line (```js) will enable JavaScript syntax highlighting on GitHub. We extracted these code blocks, along with any language data, and then filtered our dataset to contain only Node.js code. To do this, we discarded any code blocks that were not in the JavaScript language, such as bash commands used to demonstrate package installation, but kept those without a specified language, as not all READMEs utilise syntax highlighting. Then, to discard more unrelated code snippets, we filtered out common install commands such as npm install. Within Node.js repositories, it is also
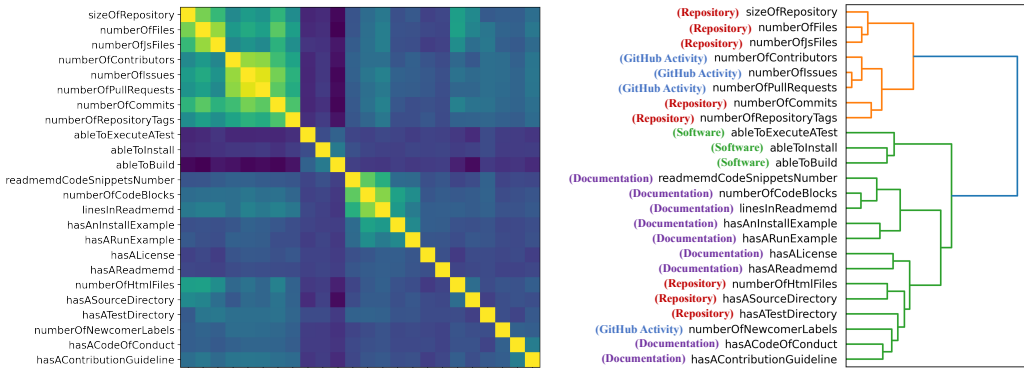
Fig. 3. Features: correlations (left) and clustering (right). Lighter fields correspond to a strong positive correlation between the features, and darker fields to a strong negative correlation. X-labels are omitted as they follow the order (optimised to co-locate correlated features) of the y-labels. The dendrogram groups correlated features closely together. Shown are the correlations based on the 11,127 data points for which all feature values are available; not shown are the four timestamp-related features.

common to see code blocks containing the results of the previous snippet, often in the form of a JSON data object or array; as such, we filtered out any singular objects or arrays from our set of snippets.

To simulate how developers include the package in their projects and test its usage, we use a similar approach to Section 4.1. Instead of building the package itself, we built the empty Node.js package, which depends on the selected npm package. After that, we executed the code snippets inside our empty package.

As shown in Table 4, we selected all 104,364 npm package repositories as the input of the executable documentation experiment. We found that 97,006 packages (92.95%) are successfully installed. From these installed packages, we found that only 64,280 packages (61.59%) have at least one code snippet in their README. In the end, we executed 220,324 code snippets and found that only 33,484 (15.20%) of them were successfully executed without any error. Note that the code snippet executing process took around 14 days of execution.

5 CORRELATING FEATURES BY PERSPECTIVE

To answer (RQ2) *What features of an npm package correlate from different perspectives?* we initially perform a descriptive analysis of all features. From this analysis, we first describe the most positive and most negative correlations in the dataset to understand the trade-off between features and perspectives. We then manually investigate correlations between different perspectives. The following analysis characterises the 11,127 packages for which all 30 feature values are available; we ignore githubLink and mostPopularFileExtension. This was necessary to achieve a consistent picture, as not all used techniques allow for missing values.

Figure 3 shows the Spearman rank-order correlation coefficients between the 28 features and clustered with Wards hierarchical clustering approach. We find that features from the same type are more likely to have strong positive correlations among them. The GitHub activity features like numberOfIssues and numberOfPullRequests have the strongest positive correlation among any feature pairs. For documentation features, numberOfCodeBlocks and linesInReadmemd have a strong positive correlation, but less than the GitHub activity features. For repository features, numberOfCommits and numberOfRepositoryTags also have a strong positive correlation and

Table 5. Top 5 negative correlations of features grouped by their types (U: user, C: contributor, B: both of them, N: none of them).

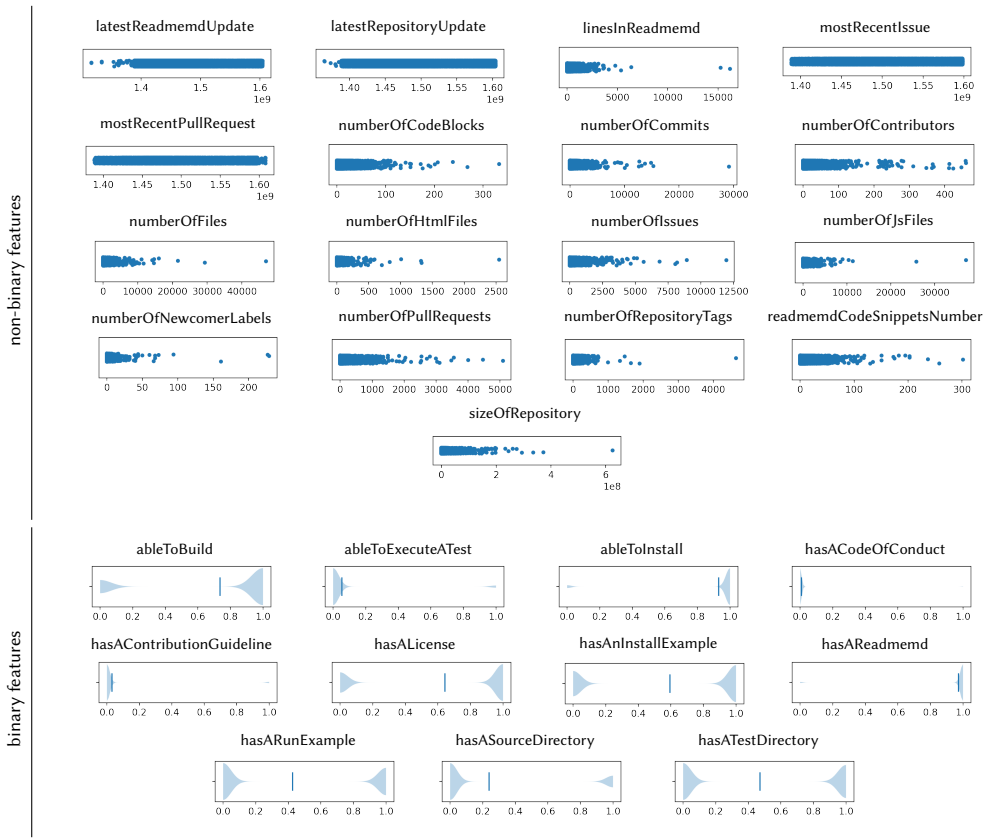| Feature type	Software feature	Correlation	Corresponding feature
Documentation	ableToBuild	-0.0962	hasAContributionGuideline (C)
		-0.0536	hasACodeOfConduct (C)
		-0.0448	linesInReadmemd (B)
		-0.0426	numberOfCodeBlocks (U)
		-0.0393	hasARunExample (U)
	ableToExecuteATest	-0.0518	hasAContributionGuideline (B)
		-0.0281	hasACodeOfConduct (C)
		-0.0199	hasARunExample (U)
		-0.0074	hasAReadmemd (B)
		-0.0065	numberOfCodeBlocks (U)
	ableToInstall	-0.0255	hasACodeOfConduct (C)
		-0.0216	hasAContributionGuideline (C)
		-0.0096	hasAnInstallExample (B)
		-0.0092	linesInReadmemd (B)
Repository	ableToBuild	-0.2485	numberOfFiles (N)
		-0.2291	numberOfCommits (N)
		-0.2224	sizeOfRepository (B)
		-0.2126	hasASourceDirectory (C)
		-0.1847	numberOfRepositoryTags (N)
	ableToExecuteATest	-0.1141	sizeOfRepository (B)
		-0.1114	numberOfCommits (N)
		-0.0997	numberOfFiles (N)
		-0.0897	hasASourceDirectory (C)
		-0.0818	numberOfRepositoryTags (N)
	ableToInstall	-0.0947	sizeOfRepository (B)
		-0.0808	numberOfFiles (N)
		-0.0683	numberOfCommits (N)
		-0.0648	numberOfJsFiles (N)
		-0.0270	numberOfRepositoryTags (N)
GitHub activity	ableToBuild	-0.2249	numberOfIssues (B)
		-0.2139	numberOfPullRequests (B)
		-0.1893	numberOfContributors (B)
		-0.0662	numberOfNewcomerLabels (C)
	ableToExecuteATest	-0.1099	numberOfIssues (B)
		-0.1064	numberOfContributors (B)
		-0.1029	numberOfPullRequests (B)
		-0.0235	numberOfNewcomerLabels (C)
	ableToInstall	-0.0484	numberOfIssues (B)
		-0.0466	numberOfContributors (B)
		-0.0379	numberOfPullRequests (B)
		-0.0274	numberOfNewcomerLabels (C)

Fig. 4. Value distribution of 28 features in alphabetical order. For non-binary features (top five rows), we use strip plots to show the distributions qualitatively by showing single data points; for binary features (bottom three rows), we use violin plots with shown means to illustrate the amount of data at either end of the distribution. Shown are all data points (11,127 to 104,364).

belong to the same cluster. Software features (i.e., ableToExecuteATest, ableToInstall, ableToBuild) have positive correlations among each other, but the correlations are not strong. On the other hand, the features from different types tend to have either small positive correlations or negative correlations.

Table 5 shows the top negative correlations between software features and other features, which are top negative correlations among any feature pairs shown in Figure 3. We find that various features that measure the size of a repository (e.g., number of files, commits, issues, pull requests, contributors) tend to be negatively correlated with the ability to install, build, and execute tests on a package. This makes intuitive sense as larger repositories are more complex and more likely to encounter issues regarding runnability. In addition, we note that correlations between several of the documentation features and runnability also tend to be slightly negative. Curiously, even the correlation between hasAnInstallExample and the ability to install a package is not positive.

Figure 4 shows the distributions of the feature values. Note that the strip plots cannot show all 11,127 data points; however, they provide a subjectively better qualitative and quantitative characterisation of the distribution than violin plots.
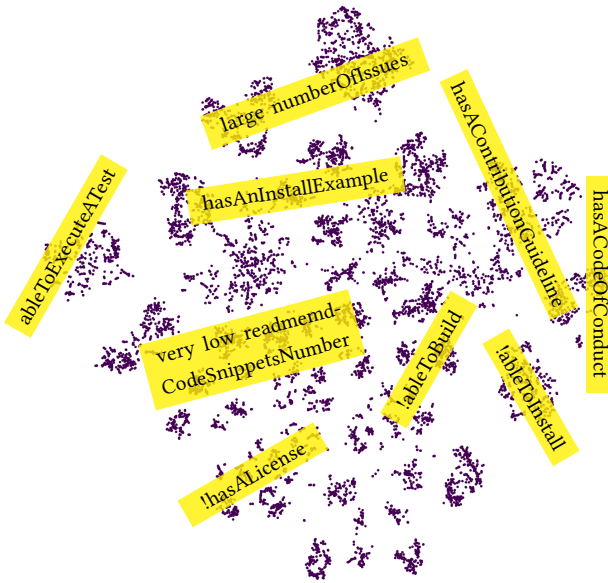
Fig. 5. Repositories in 2D. The axes do not have any particular meaning in projections like these, which is why we removed them. Note that the clusters at the very bottom end of this figure are not easily characterised by a single feature but by combinations of features.

Figure 5 shows the clusters of npm packages using the t-distributed Stochastic Neighbour Embedding (t-SNE) [62] to project the 28D data points into 2D. t-SNE's reduction process attempts to preserve the distances in the high-dimensional space as much as possible. Interestingly, a large number of clusters has formed. We have manually inspected the clusters and added labels for cases that we have found interesting. While we have to be careful not to over-interpret this reduced representation, we can observe co-located areas where the neighbourhood seems intuitively reasonable. For example, at the right, repositories are listed with contribution guidelines and codes of conduct, both of which can be qualities of mature repositories – and we can indeed find these repositories in the vicinity of those with large numbers of issues; further to the left, we can find repositories for which it is possible to execute tests. Similarly, we can find repositories that cannot be built or do not have a license in the lower part of the plot. Lastly, and a little bit to the left of the centre of the figure, we have repositories where the README.md has a small number of code snippets: this seems to place them at the (fuzzy) boundary between possible immature projects and mature ones.

Summary: There are strong positive correlations among features from the same type. Other feature combinations present trade-offs – in particular software features tend to be negatively correlated with other features.

6   PREDICTING WHETHER OR NOT AN NPM PACKAGE IS RUNNABLE

To answer (RQ3) *What features of an npm package predict whether it is runnable or not?* we first need to clarify what we mean by runnable that can be expressed by our features defined in Table 1 in our experiments. To do so, we apply the DUO principle (Data Mining Algorithms Using/Used-by
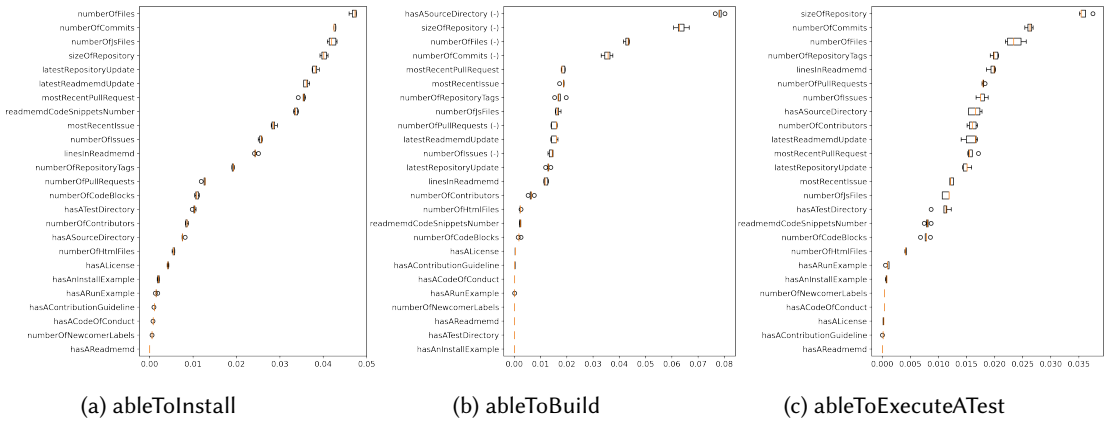
Fig. 6. Permutation importance of a good predictive model for ableToInstall, ableToBuild, and ableToExecuteATest (from left to right). Larger values mean that the prediction is more sensitive with respect to that parameter; hence it can be seen as more important. The + and − mark features with weak correlations with the respective target > 0.2 and < −0.2; almost all correlations are indeed in $[−0.2, 0.2]$, and no correlation was outside of $[−0.4, 0.4]$.

Optimizers [2]) to mine insights by using optimisers: we employ auto-sklearn [20] to automatically search over the space of machine learning models to achieve an optimised insight, i.e., to achieve a good approximation of the truth while reducing potential limitations of modelling technology and its default or manual configuration. auto-sklearn is a combination of the machine learning library sklearn [49], the algorithm configurator smac [29], and a set of preprocessing strategies for data.

We investigate three different binary target features as our interpretation of code being runnable, focusing on essential aspects of software engineering:

(1) ableToInstall: while this appears to be a weak condition, it can be a necessary condition for subsequent code development;
(2) ableToBuild: stronger than the previous one, and can indicate higher usefulness;
(3) ableToExecuteATest: if true, then this can indicate correctness with respect to the specification;

We consider all 104,364 repositories, and we consider all features as inputs except for the three ableTo* (to avoid the accidental explanation through strong correlations), as well as the two string-based features; this leaves us with 25 input features. For the model search, we perform 5-fold cross-validation. For each fold, auto-sklearn has 10 minutes allocated (single CPU core) to find a well-performing model for a target feature.[1] The average F1-scores and accuracy scores over the five folds (per target) are as follows:

(1) ableToInstall: F1=0.96, accuracy=0.92. This is not too surprising, as 92% of the data set is installable.
(2) ableToBuild: F1=0.83, accuracy=0.73. 72% can be built.
(3) ableToExecuteATest: F1=0.09, accuracy=0.95. The F1 score is very low, although this is partly due to the imbalanced dataset: it is possible to execute at least one of the tests only for 5% of the repositories.

To dig deeper into the results, we take (for each target) one of the generated models and investigate the permutation importance of the features [7]. In this post-hoc approach, a single column of the

---

[1]As a performance reference: a random forest from sklearn is typically trained in about one second on our standard laptop.

validation data is randomly shuffled, leaving the target and all other columns in place. This metric measures the effect on the accuracy of predictions in that shuffled data. Due to randomised effects, this process is repeated five times for each feature (sklearn default).

Figure 6 shows the results, where larger values indicate higher importance in these tuned models. As we can observe, to predict ableToInstall, a large number of features is necessary, with number-based and size-based features being the four most important ones. To predict ableToBuild, these features remain important, although the now most important one is hasASourceDirectory. To predict ableToExecuteATest, the previously mentioned features remain important, but they are now joined by linesInReadmemd.

Figure 6 also indicates positive and negative correlations with the respective target features. In general, all correlations with the targets are very weak, i.e., almost all correlations with the targets are within $[-0.2, 0.2]$ and are neither marked with a + or a −. ableToBuild stands out a bit as a number of features are weakly negatively correlated with it.

> **Summary:** Predicting the runnability of a package is viable (i.e., high F1 score). Repository features are particularly important for predicting the runnability.

7 DISCUSSION

We now discuss our results and provide suggestions for both researchers and practitioners.

(1) *Audience perspective matters.* As shown by our analysis of package features, not all features are relevant from all perspectives, and some are even negatively correlated. For example, the user perspective is more interested in documentation than the contributor perspective, which may be interested in workflow characteristics such as the pull-request and issue management systems. Our survey results also confirmed that package users are more focused on documentation features. For example, one survey respondent mentioned that they adopt a package due to "*Usefulness, good documentation, recent releases*". On the other hand, contributors are more focused on the contribution guidelines as mentioned by another respondent: "*simple rules for issues and pull requests*".

For practitioners, we suggest that taking into account both the user and contributor may help the overall attractiveness of their Open Source projects. Furthermore, opportunities for future work may be tool support for recommending projects based on the identified features and the different scenarios of these perspectives. Our results confirm that automatically predicting the runnability of a package may be viable. For researchers, we suggest that our perspectives open up different scenarios for the different practitioners and a re-evaluation of existing metrics as well as investigating new metrics that can capture these two perspectives. The simple heuristics of using the popularity, dependency usage, stars and downloads would need to be re-evaluated, especially for selecting representative samples for empirical studies.

(2) *Maintaining the runnability of npm packages is non-trivial.* According to our results, the definition of successfully running a project is not trivial. Although some work has looked at a single definition, no work had looked at multiple types of runnability. For practitioners, we suggest considering these different types when developing projects. This could be an issue if the project is constantly evolving, causing especially code snippets in the documentation (e.g., README) to be outdated. Furthermore, these snippets may be the usage examples or installation instructions, and therefore important for users and contributors. Another answer from a survey respondent supports this: "*One thing that indicates a good package is an example*

of someone using it to solve an existing problem. For npm packages, those are usually not found in the package's documentation, but on someone's blog".

8 THREATS TO VALIDITY

Threats to *construct validity* exist in the appropriateness of our feature list and perspectives. We mitigated these threats by conducting the survey to verify the relevance of features and perspectives for assessing the quality of packages. Our feature list is taken from related works related to npm in particular or software reuse in general; however, only a few features have been used for assessing the package quality. In this case, we received 33 responses to confirm the relevant level of features (i.e., 23 out of 30 features vote agree more than disagree).

Threats to *internal validity* involve the correctness of tools and techniques used in this study. We use multiple features extracted from GH-Node.js. The threat is that sometimes some packages do not have some features or are unavailable to extract, so we applied a filter to remove packages that have at least one missing feature for RQ2, thus making this threat minimal.

Threats to *external validity* correspond to our ability to generalise results. Because our data and conclusions are based on a large number of npm packages hosted on GitHub, we cannot generalise our results to all projects in general. Not all npm packages are hosted on GitHub, and some Node.js libraries use alternative package managers like Yarn. Our research is also focused on Node.js and npm only; there are similar package management systems for other languages, such as PyPI for Python and Maven for Java.

9 RELATED WORK

Analysis of Repositories. Recent studies have explored dependency networks in different aspects. Some studies investigated the structure and evolution of the dependency networks and revealed their issues such as dependency hell and technical lag [15, 33, 67]. Several studies focused on how to detect known security vulnerabilities from third-party libraries in the applications [8, 66]. Cogo et al. [9] performed an empirical study of dependency downgrades and found that downgrades occur because developers want to avoid some defects from a specific version and some incompatibility issues. Hejderup et al. [25] extracted the call graph for software to build a fine-grained representation of the dependency network.

Social coding in the ecosystem is also an emerging research area. Some studies focused on how developers have social interactions with each other [10, 11, 14, 46]. The relationship between different groups of developers in coding collaboration is explored in various ecosystems [19, 23, 46]. Qiu et al. [52] focused on how social coding impacts the chances of long-term engagement. Work such as Blincoe et al. [6] looked at the reference coupling between projects in the ecosystem. Other studies have shown that the maintenance of these dependencies is critical to the ecosystem [16, 35, 40, 66].

Code Snippet Executability. Documentation is an important part of choosing a library [36] and the popularity of GitHub repositories [1], and example code is in itself an important aspect of good documentation. Code snippets are primarily used online in documentation, tutorials, and collaborative sites like Stack Overflow to demonstrate how software and APIs should be used; however, not all code snippets are usable as-is [26, 39, 53, 65]. For developers learning APIs, insufficient or inadequate examples can be a major obstacle [54], and many code snippets online are incomplete, contain errors, or simply do not work. In many cases, code snippets become outdated as software evolves, and despite being the first resource many developers will see, official software documentation is frequently out-of-date, and changes to the software are not immediately reflected in documentation [37]. There is also little support for developers writing documentation,

which makes maintaining up-to-date, executable code snippets in GitHub repositories a non-trivial task [12].

Studies of the executability of online code snippets show that most are not executable; a study of online coding tutorials found that only 26% of code snippets could be executed successfully, and no tutorial could be executed to completion [39]. Gistable [26], a framework for running Python code snippets found on GitHub using the gist system, found that only 25% of code snippets were executable by default. These numbers appear to be consistent with other research into code snippet executability [27, 50, 65], with some variance depending on the language.

10 CONCLUSION AND FUTURE DIRECTIONS

With more than 1.7 million packages to choose from, selecting a suitable npm package is a difficult task – and the factors that influence this decision vary based on the developer intentions. Most existing research into library selection has focused on the perspective of a user of a package; in this paper, we have investigated the perspective of user and contributor.

In this study, we first conducted a survey with 33 respondents to understand which npm package features do practitioners find relevant for assessing package quality. We found that users and contributors of npm packages share similar views when choosing a package, as half of the considered features belong to both perspectives. However, the user perspective is more focused on the documentation and the usage of the package. The contributor perspective, on the other hand, is more focused on contribution guidelines. Interestingly, developers believe that most repository features do not belong to any perspectives and do not reflect the quality of packages.

We then created the GH-Node.js dataset, a dataset containing a curated snapshot of npm packages that the research community can utilise. We identified a set of 30 features important to one or both perspectives and identified correlations between different perspectives. We found that features from different perspectives are not necessarily correlated; in fact, in some cases, negatively correlated. This suggests that our different perspectives are important and that trade-offs exist; users and contributors have different priorities. This also suggests a need for new metrics to capture these perspectives, which complement the existing features such as popularity, dependency usage, stars, and download count.

We also evaluated the runnability of packages in terms of the runnability of test cases and example code snippets. Out of 11,127 npm packages that were updated in 2020, we find that 8,199 (73.68%) are able to be built, and only 607 (7.40%) are able to pass all test cases. Out of 104,364 npm packages from our dataset, we found that 97,006 (92.95%) could be successfully installed, 64,280 (61.59%) have at least one code snippet in their README, and out of 220,324 code snippets, 33,484 (15.20%) were able to execute successfully. Using this data, we investigated if we could predict the runnability of a package. We find that predicting a package runnability depended heavily on what metric of runnability we use; for example, we find that there is a strong correlation between the ability to install a package and its number of files and commits; however, these features are less important in predicting the ability to build a package.

Our work lays the groundwork for future work on understanding how users and contributors select appropriate npm packages. We suggest that practitioners should take package audiences into account to help the attractiveness of their packages. Package owners and contributors should maintain the runnability of their package for attracting and helping new users and newcomer contributors since this is not trivial. Potential future avenues for researchers include (1) a package recommendation system based on the runnability of packages and (2) an exploration of new metrics that can measure the package quality with the consideration of both user and contributor perspectives.

ACKNOWLEDGMENT

REFERENCES

[1] K. Aggarwal, A. Hindle, and E. Stroulia. Co-evolution of project documentation and popularity within github. In IEEE/ACM Mining Software Repositories Conference (MSR), pages 360–363, 2014.

[2] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu. Better software analytics via "duo": Data mining algorithms using/used-by optimizers. Empirical Software Engineering (ESME), 25(3):2099–2136, 2020.

[3] D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye. Do software developers understand open source licenses? In International Conference on Program Comprehension (ICPC), May 2017.

[4] M. Beller, G. Gousios, and A. Zaidman. TravisTorrent: Synthesizing travis CI and GitHub for full-stack research on continuous integration. In IEEE/ACM Mining Software Repositories Conference (MSR), May 2017.

[5] J. Bennett. Choosing a javascript library. https://www.b-list.org/weblog/2007/jan/22/choosing-javascript-library/. (Accessed on 01/13/2021).

[6] K. Blincoe, F. Harrison, N. Kaur, and D. Damian. Reference coupling: An exploration of inter-project technical dependencies and their characteristics within large software ecosystems. Information and Software Technology (IST), 110:174–189, 2019.

[7] L. Breiman. Random forests. Machine Learning, 45(1):5, 2001.

[8] B. Chinthanet, S. E. Ponta, H. Plate, A. Sabetta, R. G. Kula, T. Ishio, and K. Matsumoto. Code-based vulnerability detection in node.js applications: How far are we? In IEEE/ACM International Conference on Automated Software Engineering (ASE), Sept 2018.

[9] F. R. Cogo, G. A. Oliva, and A. E. Hassan. An empirical study of dependency downgrades in the npm ecosystem. IEEE Transactions on Software Engineering (TSE), pages 1–1, 2019.

[10] E. Constantinou and T. Mens. An empirical comparison of developer retention in the RubyGems and npm software ecosystems. Innovations in Systems and Software Engineering (ISSE), 13(2-3):101–115, 2017.

[11] E. Constantinou and T. Mens. Socio-technical evolution of the ruby ecosystem in github. In International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 34–44, 2017.

[12] B. Dagenais and M. P. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In International Symposium on the Foundations of Software Engineering (FSE), pages 127–136, 2010.

[13] F. L. de la Mora and S. Nadi. Which library should I use? A metric-based comparison of software libraries . In International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pages 37–40, may 2018.

[14] C. R. de Souza, F. Figueira Filho, M. Miranda, R. P. Ferreira, C. Treude, and L. Singer. The social side of software platform ecosystems. In Conference on Human Factors in Computing Systems (CHI), page 3204–3214, 2016.

[15] A. Decan, T. Mens, and M. Claes. An empirical comparison of dependency issues in OSS packaging ecosystems. In International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 2–12, feb 2017.

[16] A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In IEEE/ACM Mining Software Repositories Conference (MSR), pages 181–191, 2018.

[17] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. In ACM SIGSAC Conference on Computer and Communications Security, pages 2187–2200, oct 2017.

[18] O. Elazhary, M.-A. Storey, N. Ernst, and A. Zaidman. Do as i do, not as i say: Do contribution guidelines match the GitHub contribution process? In IEEE International Conference on Software Maintenance

and Evolution (ICSME), Sept. 2019.

[19] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry. The Robot Operating System: Package reuse and community dynamics. Journal of Systems and Software (JSS), pages 226–242, 2019. ISSN 01641212.

[20] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: The next generation. Computing Research Repository (CoRR), abs/2007.04074, 2020.

[21] K. Gallaba, C. Macho, M. Pinzger, and S. McIntosh. Noise and heterogeneity in historical build data: an empirical study of travis CI. In IEEE/ACM International Conference on Automated Software Engineering (ASE), Sept. 2018.

[22] G. Gousios. The ghtorrent dataset and tool suite. In IEEE/ACM Mining Software Repositories Conference (MSR), pages 233–236, 2013.

[23] H. Guercio, V. Stroele, J. M. N. David, R. Braga, and F. Campos. Complex network analysis in a software ecosystem: Studying the eclipse community. In International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 618–623, 2018.

[24] F. Hassan and X. Wang. Mining readme files to support automatic building of java projects in software repositories. In International Conference on Software Engineering Companion (ICSE-C), May 2017.

[25] J. Hejderup, A. van Deursen, and G. Gousios. Software ecosystem call graph for dependency management. In International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), page 101–104, 2018.

[26] E. Horton and C. Parnin. Gistable: Evaluating the executability of python code snippets on github. In IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 217–227, 2018.

[27] M. M. Hossain, N. Mahmoudi, C. Lin, H. Khazaei, and A. Hindle. Executability of python snippets in stack overflow. arXiv preprint arXiv:1907.04908, 2019.

[28] K. Huang, B. Chen, B. Shi, Y. Wang, C. Xu, and X. Peng. Interactive, effort-aware library version harmonization. In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pages 518–529, nov 2020.

[29] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In Learning and Intelligent Optimization (LION), pages 507–523, Berlin, Heidelberg, 2011. ISBN 978-3-642-25566-3.

[30] S. Ikeda, A. Ihara, R. G. Kula, and K. Matsumoto. An empirical study of README contents for JavaScript packages. IEICE Transactions on Information and Systems, E102.D(2):280–288, Feb. 2019.

[31] S. D. Joshi and S. Chimalakonda. RapidRelease - a dataset of projects and issues on github with rapid releases. In IEEE/ACM Mining Software Repositories Conference (MSR), May 2019.

[32] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining GitHub. In IEEE/ACM Mining Software Repositories Conference (MSR), 2014.

[33] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl. Structure and Evolution of Package Dependency Networks. In IEEE/ACM Mining Software Repositories Conference (MSR), pages 102–112, 2017.

[34] N. Kobayakawa and K. Yoshida. How GitHub contributing.md contributes to contributors. In Computer Software and Applications Conference (COMPSAC), July 2017.

[35] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? Empirical Software Engineering (ESME), 23(1):384–417, Feb. 2018.

[36] E. Larios Vargas, M. Aniche, C. Treude, M. Bruntink, and G. Gousios. Selecting third-party libraries: the practitioners' perspective. In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pages 245–256, nov 2020. doi: 10.1145/3368089. 3409711.

[37] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: The state of the practice. IEEE Software, 20(6):35–39, 2003.

[38] C. Macho, S. McIntosh, and M. Pinzger. Automatically repairing dependency-related build breakage. In International Conference on Software Analysis, Evolution and Reengineering (SANER), Mar. 2018.

[39] S. Mirhosseini and C. Parnin. Docable: Evaluating the executability of software tutorials. In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), page 375–385, 2020.

[40] S. Mujahid, R. Abdalkareem, E. Shihab, and S. McIntosh. Using others' tests to avoid breaking updates. In IEEE/ACM Mining Software Repositories Conference (MSR), 2020.

[41] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating GitHub for engineered software projects. Empirical Software Engineering (ESME), 22(6):3219–3253, 2017.

[42] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta. CrossRec: Supporting software developers by recommending third-party libraries. Journal of Systems and Software (JSS), 161:110460, mar 2020.

[43] E. Noei, M. D. Syer, Y. Zou, A. E. Hassan, and I. Keivanloo. A study of the relation of mobile device attributes with the user-perceived quality of android apps. Empirical Software Engineering (ESME), 22 (6):3088–3116, Mar. 2017.

[44] NPM. npm. https://www.npmjs.com/, 2010. (Accessed on 05/13/2021).

[45] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue. Search-based software library recommendation using multi-objective optimization. Information and Software Technology, 83:55–75, mar 2017.

[46] M. Palyart, G. C. Murphy, and V. Masrani. A study of social interactions in open source component use. IEEE Transactions on Software Engineering (TSE), 44(12):1132–1145, Dec 2018.

[47] A. Pano, D. Graziotin, and P. Abrahamsson. Factors and actors leading to the adoption of a JavaScript framework. Empirical Software Engineering (ESME), 23(6):3503–3534, dec 2018.

[48] J. Patra, P. N. Dixit, and M. Pradel. ConflictJS: finding and understanding conflicts between JavaScript libraries. In International Conference on Software Engineering (ICSE), pages 741–751, may 2018.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[50] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In IEEE/ACM Mining Software Repositories Conference (MSR), pages 507–517, 2019.

[51] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. Categorizing the content of GitHub README files. Empirical Software Engineering (ESME), 24(3):1296–1327, Oct. 2018.

[52] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu. Going farther together: The impact of social capital on sustained participation in open source. In International Conference on Software Engineering (ICSE), pages 688–699, 2019.

[53] B. Reid, C. Treude, and M. Wagner. Optimising the fit of stack overflow code snippets into existing code. In Genetic and Evolutionary Computation Conference (GECCO), page 1945–1953, 2020.

[54] M. P. Robillard. What makes apis hard to learn? answers from developers. IEEE Software, 26(6):27–34, 2009.

[55] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo. Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software (JSS), 145(October 2017): 164–179, 2018. ISSN 01641212.

[56] D. Sholler, I. Steinmacher, D. Ford, M. Averick, M. Hoye, and G. Wilson. Ten simple rules for helping newcomers become contributors to open projects. PLOS Computational Biology, 15(9):e1007296, Sept. 2019.

[57] I. Steinmacher, M. A. Gerosa, and D. F. Redmiles. Attracting, onboarding, and retaining newcomer developers in open source software projects. In ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW), number February, pages 1–4, 2014.

[58] X. Tan, M. Zhou, and Z. Sun. A first look at good first issues on GitHub. In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pages 398–409, nov 2020.

[59] F. Thung, D. Lo, and J. Lawall. Automated library recommendation. In Working Conference on Reverse Engineering (WCRE), number October, pages 182–191, oct 2013.

[60] Tidelift. Libraries.io - The Open Source Discovery Service. https://libraries.io/, 2017. (Accessed on 01/13/2021).

[61] P. Tourani, B. Adams, and A. Serebrenik. Code of conduct in open source projects. In International Conference on Software Analysis, Evolution and Reengineering (SANER), Feb. 2017.

[62] L. van der Maaten and G. Hinton. Visualizing data using t-sne. Journal of Machine Learning Research, 9: 2579–2605, Nov. 2008. ISSN 1533-7928 (electronic); 1532-4435 (paper).

[63] Y. Wang, M. Wen, Z. Liu, R. Wu, R. Wang, B. Yang, H. Yu, Z. Zhu, and S.-C. Cheung. Do the dependency conflicts in my project matter? In ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pages 319–330, oct 2018.

[64] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? Empirical Software Engineering (ESME), 22(6):3149–3185, Dec. 2017.

[65] D. Yang, A. Hussain, and C. V. Lopes. From query to usable code: An analysis of stack overflow code snippets. In IEEE/ACM Mining Software Repositories Conference (MSR), page 391–402, 2016. ISBN 9781450341868.

[66] R. E. Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto, and A. Ihara. Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm javascript packages. In IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 559–563, 2018.

[67] A. Zerouali, E. Constantinou, T. Mens, G. Robles, and J. Gonzalez-Barahona. An empirical analysis of technical lag in npm package dependencies. In International Conference on Software Reuse (ICSR), pages 95–110, 2018.

[68] J. Zhou, S. Wang, C.-P. Bezemer, Y. Zou, and A. E. Hassan. Studying the association between bountysource bounties and the issue-addressing likelihood of GitHub issue reports. IEEE Transactions on Software Engineering (TSE), pages 1–1, 2020.

[69] M. Zimmermann, C. A. Staicu, C. Tenny, and M. Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In USENIX Security Symposium, number February, 2019.