The Journal of Systems & Software 198 (2023) 111588

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

What are the characteristics of highly-selected packages? A case study on the npm ecosystem ☆

Suhaib Mujahid [a,b,*], Rabe Abdalkareem [c], Emad Shihab [a]

[a] Data-driven Analysis of Software (DAS) Lab, Concordia University, Montreal, Canada
[b] Mozilla Corporation, Montreal, Canada
[c] Department of Computer Science, Faculty of Science, Omar Al-Mukhtar University, Libya

ARTICLE INFO

Article history:
Received 23 March 2022
Received in revised form 2 December 2022
Accepted 5 December 2022
Available online 7 December 2022

Keywords:
highly-selected packages
Package quality
Software ecosystem
npm

ABSTRACT

With the popularity of software ecosystems, the number of open source components (known as packages) has grown rapidly. Identifying high-quality and well-maintained packages from a large pool of packages to depend on is a basic and important problem, as it is beneficial for various applications, such as package recommendation and package search. However, no systematic and comprehensive work focuses on addressing this problem except in online discussions or informal literature and interviews. To fill this gap, in this paper, we conducted a mixed qualitative and quantitative analysis to understand how developers identify and select relevant open source packages. In particular, we started by surveying 118 JavaScript developers from the npm ecosystem to qualitatively understand the factors that make a package to be highly-selected within the npm ecosystem. The survey results showed that JavaScript developers believe that highly-selected packages are well-documented, receive a high number of stars on GitHub, have a large number of downloads, and do not suffer from vulnerabilities. Then, we conducted an experiment to quantitatively validate the developers' perception of the factors that make a highly-selected package. In this analysis, we collected and mined historical data from 2,527 packages divided into highly-selected and not highly-selected packages. For each package in the dataset, we collected quantitative data to present the factors studied in the developers' survey. Next, we used regression analysis to quantitatively investigate which of the studied factors are the most important. Our regression analysis complements our survey results about highly-selected packages. In particular, the results showed that highly-selected packages tend to be correlated by the number of downloads, stars, and how large the package's readme file is.

1. Introduction

A software ecosystem is a collection of interdependent software packages that are developed and evolve together in a common technological platform (e.g., Node.js) (Serebrenik and Mens, 2015). In recent years, the proliferation of software ecosystems has led to a vast and rapid growth of the number of open-source packages.[1] For example, as of January 2022, there were over a million packages available on the registry of the Node Package Manager (npm), one of the largest software ecosystems. Furthermore, the number of packages grew by around 60% between January 2019 and January 2022 (DeBill, 2022). With the massive number of packages out there, finding the right package to use can be challenging, considering that many packages provide similar functionalities. However, there are some packages that stand out and experience high interest from developers. We believe that understanding the characteristics of these highly-selected packages is crucial since it helps developers answer the essential question: which packages a developer should select among many existing options. In addition, such understanding can help to improve the performance of package recommendation systems (Zheng et al., 2011; de la Mora and Nadi, 2018; StackOverflow, 2017; Semeteys, 2008) and enhance the user experience of package search engines (Abdellatif et al., 2020; StackOverflow, 2017; Cruz and Duarte, 2018). Furthermore, for package developers, understanding the characteristics of highly-selected packages can be helpful for various purposes, such as improving their packages to meet the requirements and experiences that users look for, acquiring the community's attention, and eventually increasing the package usage and overall reputation. Those potential implications of understanding the factors of highly-selected motivate use to study them.

☆ Editor: Fabio Palomba.

* Corresponding author.
E-mail addresses: s_mujahi@encs.concordia.ca (S. Mujahid), rabe.abdalkareem@omu.edu.ly (R. Abdalkareem), emad.shihab@concordia.ca (E. Shihab).

[1] In this paper, we use the term package referring to open source components published on software ecosystems.

https://doi.org/10.1016/j.jss.2022.111588
0164-1212/

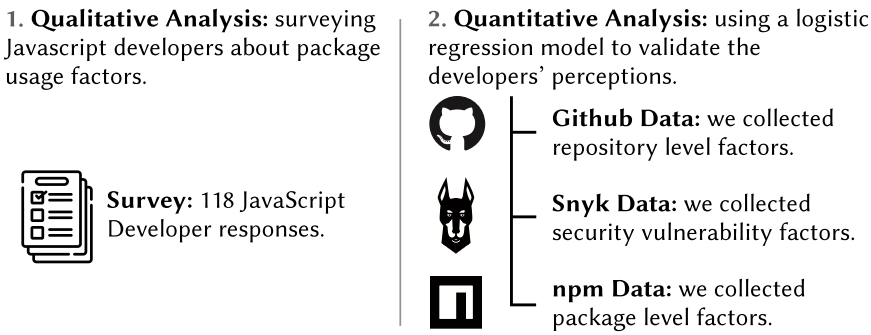

Fig. 1. An overview of our study design.

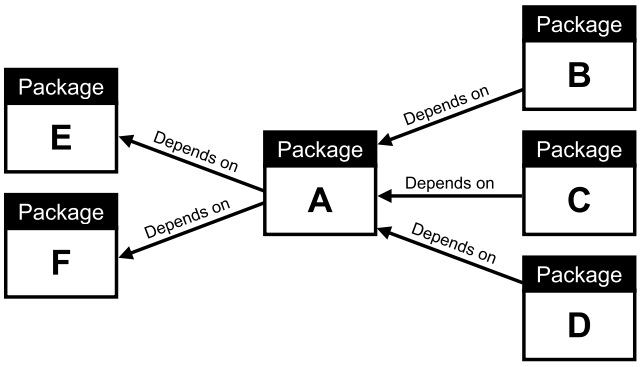

Fig. 2. Example of a dependency graph. The packages B, C and D are direct dependents of package A. The packages E and F are direct dependencies of package A.

Previous studies examined different aspects of packages published in software ecosystems, such as centrality and popularity (Abdalkareem et al., 2020; Abdellatif et al., 2020; Larios Vargas et al., 2020; Qiu et al., 2018; Mujahid et al., 2021), and a few examined the selection factors of relevant packages (Larios Vargas et al., 2020; Jadhav and Sonar, 2009). The main limitation of prior works is that they were based on a purely quantitative analysis of popular packages or only interviewing developers in a specific industrial context. That said, understanding the characteristics of highly-selected packages is still the subject of much discussion and refinement. This is because several facts include personality aspects and examining different data modalities from several sources, and a developer is typically a familiar user of a specific package. Thus, in this paper, we divided the study into two parts — qualitative and quantitative (John et al., 2000). Fig. 1 shows an overview of our study design. In the first part (referred from now on as *qualitative analysis*), we conducted a user study survey that involves 118 JavaScript developers. We asked our survey participants to fill in a form composed of 17 statements about factors they use when selecting npm packages. Then, we qualitatively analyzed the developers answers to the 17 questions using descriptive statistics.

In order to provide validation to the findings of the qualitative analysis, we conducted *quantitative analysis* on a set of 2527 npm packages grouped into highly-selected and not highly-selected packages. Similar to prior work (Bavota et al., 2015; Lee et al., 2020; Tian et al., 2015), we estimated the highly-selected packages based on the number of direct dependent packages (illustrated in Fig. 2). Then, we mined and analyzed the selected packages and collected quantitative data to present the factors studied in our survey. Next, we used regression analysis to quantitatively investigate which of the studied factors are the most important for developers to select a package to use.

The survey results showed that JavaScript developers believe that when selecting a package to use, they look for packages that are: well-documented, receive a high number of stars on GitHub, have a large number of downloads, and do not suffer from security vulnerabilities. Moreover, our regression analysis complemented our survey results about highly-selected packages. For example, our developers' survey and regression analysis revealed that developers select packages with a high number of stars and downloads. Also, it described the differences between the developers' perceptions about highly-selected packages and their characteristics. In general, our work makes the following key contributions:

- We performed a mixed qualitative and quantitative analysis to investigate the characteristics of highly-selected packages on the npm ecosystem. We presented our results from surveying 118 JavaScript developers and validated the survey results through a quantitative analysis of 2427 npm packages.
- We identified the most important factors that packages' users should consider when selecting an npm package to use in their projects.
- We also provided practical implications for packages' maintainers, the npm ecosystem's maintainers, and researchers and outline future research avenues.
- To support replicating our work and future research, we make our dataset publicly available (Mujahid et al., 2022).

The remainder of this paper is structured as follows. In Section 2, we present the work that is related to our study. Section 3 describes the study design and presents the results of the qualitative analysis. Section 4 describes the study design and presents the results of the quantitative analysis. We discuss the implications of our study in Section 5. We discuss the threats that may affect the validity of the results in Section 6. Finally, Section 7 concludes our work.

2. Related work

The increasing trend of depending on software ecosystems by developers has motivated researchers to understand the developer's perspective about using third-party packages.

Haefliger et al. (2008) studied the reuse pattern and practices in open source applications. Their study showed that experienced developers reuse more code than less experienced developers. Abdalkareem et al. (2017), Abdalkareem (2017) studied an emerging code reuse practice in the form of lightweight packages in the software ecosystem. Their study was conducted to understand why developers use trivial packages. Their results showed that these packages are prevalent in PyPI (Python Package Index), but 70.3% of the developers considered using these packages is a bad practice. In addition, Xu et al. (2019) studied the

reason behind the reusing and re-implementing of external packages in software applications. They found that developers often replace their self-implemented methods with external libraries because they were initially unaware of the library or it was unavailable back then. Later on, when they became aware of a well-maintained and tested package, they replace their own code with that package. Although developers preferred to reuse code than re-implement it, they replaced an external heavy package with their implementation when they believed that they were only using a small part of its functionalities or if it became deprecated. Haenni et al. (2013) conducted a survey with developers about their decision-making while introducing a dependency to their applications. Surprisingly, the study found that developers generally do not apply rationale while selecting the packages; they used any package that accomplishes the required tasks.

Some other work focused on the health of open source software ecosystems. For example, Zerouali et al. (2019a) studied the impact of outdated and vulnerable Javascript packages and highlighted the risk of potential security vulnerabilities. Another study by Decan et al. (2017) performed an empirical comparison of dependency issues in open source software ecosystems. Their results motivate the need for improvements to handle issues related to adopting and updating package dependencies.

Other work also focused on examining the popularity growth of packages within an ecosystem. For example, Qiu et al. (2018) studied the growth of popular npm package. Their finding showed that lifetime, number of dependents, and added new functionalities play significant roles in popularity growth. Chatzidimitriou et al. (2019) used network analysis and information retrieval techniques to study the dependencies that co-occur in the development of npm packages. Then, they used the constructed network to identify the communities that have been evolved as the main drivers for npm's exponential growth. Their findings showed that several clusters of packages can be identified. Zerouali et al. (2019b) examined a large number of npm packages by extracting nine popularity metrics. They focused on understanding the relationship between the popularity metrics. They found that the studied popularity metrics were not strongly correlated.

Other work focused on understanding the process used by developers to select packages and attempted to provide some guidelines. Pano et al. (2018) focused on understanding factors that developers look for when selecting a JavaScript framework (e.g., React). Based on interviewing 18 decision-makers, they observed a list of factors when choosing a new JavaScript framework, including the community's size behind the framework. del Bianco et al. (2011) provided a list of factors that influence the trustworthiness of open source software components. Their list had five categories, including quality and economic categories. Also, in their study, Hauge et al. (2009) observed that many organizations apply informal selection process based on previous experience, recommendations from experts, and information available on the Internet. Franch and Carvallo (2003) investigated to adapt the ISO quality model and assign metrics to be used as a measure for selecting software components. Their study suggested that relationships between quality entities need to describe explicitly.

The main goal of our study is to examine the characteristics of highly-selected packages within the npm ecosystem. In many ways, our study is complementary to prior work since we focus on understanding factors that make a package highly-selected. That said, our study is one of the only studies to use mixed research methods, which provide us with more complete and synergistic utilization of data than any separate quantitative and qualitative data collection and analysis.

3. Qualitative analysis

This analysis aims to survey JavaScript developers to understand the characteristics of packages that JavaScript developers look for when selecting an npm package to use. In this study, we surveyed 118 JavaScript developers.

3.1. Study design

This section presents our survey design, participant recruitment, and data analysis methods.

3.1.1. Survey design

To understand which factors developers look for when selecting an npm package, we designed a survey containing three main parts. The first part contained questions related to the background of the participants. We asked these questions to ensure that our survey participants have sufficient experience in software development and in selecting and using npm packages. In this part, we asked the following questions:

1. How would you best describe yourself? A question with the following choices and the last choice is a free-text form: Full-time, Part-time, Free-lancer, and Other.
2. For how long have you been developing software? A selection question with the following options: <1 year, 1–3, 4–5, more than 5 years.
3. How many years of JavaScript development experience do you have? A selection question with the following options: <1 year, 1–3, 4–5, more than 5 years.
4. How many years of experience do you have using the Node Package Manager (npm)? A selection question with the following options: <1 year, 1–3, 4–5, more than 5 years.
5. How often do you search for npm packages? A question with the following options: Never, Rarely (e.g., once a year), Sometimes (e.g., once a month), Often (e.g., once a week), Very often (e.g., everyday).
6. Which search engine interface do you use to find relevant npm packages? A question with the following multiple choices and the last choice is a free-text form: Online search on the npm web page (i.e., *npms*), Command line search, Google or other general web search engines, and Other.

In the second part of the survey, we had a list of statements that present seventeen factors that can affect selecting npm packages. In particular, we asked the question *"How important are the following factors when selecting a relevant npm package?"* Table 1 reports the seventeen factors statements. For each statement, the table presents each factor's definition and the rationale behind asking about it. In the survey, we asked participants to rate these statements using a Likert-scale ranges from 1 = not important to 5 = very important (Oppenheim, 1992). We chose to investigate these factors for two main reasons. First, our literature review (Mujahid, 2021) indicated that these factors are known to impact the use and selection of npm packages. Second, we focused on studying factors that developers can easily observe through examining the package source code or its software repository, e.g., from the GitHub website.

In the last part of our survey, we asked the participants an open-ended question about whether they had any additional comments or other factors that they look for when they select a package. We asked this open-ended question to give our survey participants maximum flexibility to express their opinion and experience with the selection of npm packages, which also complies with the survey design guidelines (Dillman, 2011).

Table 1
List of factors used in selecting a packages from the npm ecosystem.

Factor	The survey statements	Rationale
Forks	The number of forks for the package's source code on GitHub.	The number of forks that packages receive indicates that the packages are active, and many developers are contributing to these packages (Gousios et al., 2014).
Watchers	The number of watchers of the package's GitHub repository.	Developers can watch package repositories on GitHub so they can receive notifications about package development activities (Sheoran et al., 2014). The higher the number of watchers on a package indicates that the package is well-known and used by many developers. We consider that packages with an increased number of watchers refer to highly-selected packages.
Contributors	The number of contributors to a package's GitHub repository.	The higher the number of contributors to a package repository shows that the package is more likely to attract developers (Yamashita et al., 2016).
Downloads	The number of downloads the package has.	The packages that have a higher number of downloads indicate that the packages are highly selected and used (Abdellatif et al., 2020).
Stars	The number of stars of a packages on GitHub.	A high number of stars that a npm package receives on GitHub could indicate to developers that the package is more likely popular, which may attract them to use the package (Borges and Valente, 2018; Dabbish et al., 2012).
Dependencies	The number of dependencies the npm package has.	A larger number of dependencies could not attract more developers to use the packages since prior work shows that packages with a more considerable dependency may lead to dependency hell (Abdalkareem et al., 2017).
License	Wether the npm package has a permissive or restrictive software license.	When evaluating a package, it is also essential to consider non-functional requirements, such as the license. Using a package with no license or with a license that does not match the developer organization's usage and policies can quickly become a problem (Meloca et al., 2018; Team, 2019).
Documentation	Whether the npm package repository has online documentation, e.g. README file.	The package that is well-documented and has an organized README file is more likely to be used by many developers (Begel et al., 2013; Hata et al., 2015).
Test code	Whether the npm package has test cases written.	Packages that have test code written are more likely to attract developers to use them since it indicates that the packages are well-tested (Abdalkareem et al., 2017).
Build status	The build status of the npm package for example from Travis CI.	The presence of a high number of failed builds in the package repository may lead developers not to use the package (Abdellatif et al., 2020).
Vulnerabilities	If the npm package depends on vulnerable dependencies.	If a npm package is affected by vulnerabilities, it may concern developers and deter them from selecting and using the package (Abdellatif et al., 2020; Abdalkareem et al., 2020).
Badges	If the package repository has badges.	The presence of badges in the package repository indicates that the package is of good quality that attracts developers to use the package (Trockman et al., 2018).
Website	If the package has a custom website.	The presence of a website for the package indicates that the package is supported by an organization, which is usually a signal that there is more than one maintainers or major contributor (i.e., there is support by an organization) (Qiu et al., 2019).
Releases	The release frequency of the package.	A package with several releases indicates that the package is well maintained.
Closed issues	The number of closed issues in the package's repository.	The number of closed issues indicates how well-maintained the package is and reveals how maintainers of the package respond to issues. Packages with a large percentage of closed issues attract more developers to use the package (Abdellatif et al., 2020).
Commit frequency	The commit frequency in the package repository.	Developers mainly look for well-maintained and active packages to use. Prior work also shows that the number of commits a package receives can give a good indication of how active the package is, which results in high selection (Abdellatif et al., 2020).
Usage	The number of projects using the package on GitHub.	Packages that are used by many other developers are more likely to attract more developers to use (Abdalkareem et al., 2020).

Once we had our survey questions, we shared the survey with three of our colleagues who are experts in JavaScript programming and using packages from npm. We performed this pilot survey to discover potential misunderstandings or unexpected questions early on and improve our survey (Dillman, 2011).

3.1.2. Participant recruitment

To identify the participants in our survey, we needed to reach out to developers who are the experts in selecting and using JavaScript packages. Thus, we resorted to the public registry of npm (npm, 2017). The registry contains information on each package published on npm, including information about the developers maintaining the package. We used the npm registry to collect a list of emails and names of JavaScript developers who selected and used a sufficient number of npm packages. To do so, we analyzed the npm registry, and for each package, we extracted and counted the number of its dependencies and the contacts information of the developers who are maintaining the package. Then, we selected the top thousand developers based on the number of their distinct package dependencies. It is important to note that we selected developers who use a large number of packages since they likely went through the process of selecting npm packages several times.

Once we identified this initial sample of developers, we examined all the names and email addresses of the identified developers to exclude duplicated emails and names. Based on this step, we identified 931 unique JavaScript developers. Next, we sent email invitations of our survey to the 931 unique developers. However, since some of the emails were returned for several reasons (e.g., invalid emails), we successfully reached 895 developers. In the end, we received 118 responses for our survey after having the survey available online for ten days. This number of responses translates to a 13.18% response rate, which is comparable to the response rate reported in other software engineering surveys (Smith et al., 2013).

3.1.3. Survey participants

Table 2 shows the positions of the participants, the development experience of the participants, the JavaScript experience of the participants, and their experiences in using npm ecosystem.

Table 2
Participants' position, and their experience in software development, JavaScript, and use of the npm package manager platform.

Developers' position	Occurrences	Development experience	Occurrences	Experience in JavaScript	Occurrences	Experience in using npm	Occurrences
Full-time	84	<1	2	<1	0	<1	0
Part-time	9	1–3	21	1–3	25	1–3	59
Freelancer	15	4–5	15	4–5	37	4–5	20
Other	10	>5	80	>5	56	>5	39

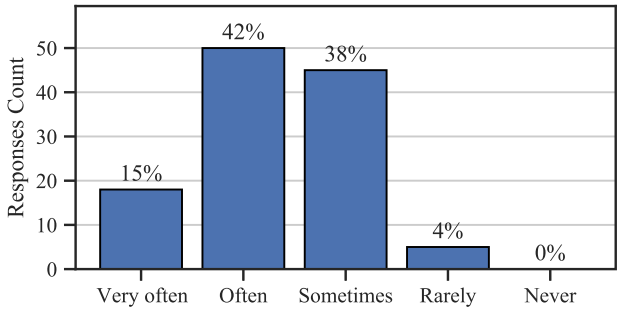

Fig. 3. Survey responses regarding how often our survey participants search for npm packages. In our survey, the question has the following answers: never, rarely (e.g., once a year), sometimes (e.g., once a month), often (e.g., once a week), very often (e.g., everyday).

As for the participants' positions, 84 participants identified themselves as full-time developers and 9 participants as part-time developers. Interestingly, 15 participants identified themselves as freelancers. The remaining ten participants identified themselves as having other positions not listed in the question, including open-source developers, IT specialists, and Ph.D. students.

Of the 118 participants in our survey, 80 participants have more than 5 years of development experience and 15 responses have between 4 to 5 years. Also, 21 participants claimed to have between 1 to 3 years of experience, and only two participants have less than one year of development experience. In addition, 56 participants have more than 5 years of experience in JavaScript, 37 participants have experience in using JavaScript between 4 to 5 years, and 25 participants claimed to have between 1 to 3 years of experience.

We also asked our survey participants about their experience in using packages from the npm ecosystem. The majority of our survey participants indicated that they have more than one year of experience using npm. Specifically, 39 participants have more than 5 years of experience using npm and 20 responses have between 4 to 5 years. Finally, 59 participants claimed to have between 1 to 3 years of experience.

In addition, to inquire our survey participants about their development experience, we asked them how often they search for npm packages and which search engine they used to perform their search. Fig. 3 reports the result related to participants' habits about how often they search for npm packages. Of the 118 participants, 15% indicated that they search for npm packages very often, and 42% indicated that they often search for new npm packages. Almost all the remaining participants (38%) indicated they sometimes look for npm packages. Interestingly, only 4% of our survey participants reported that they rarely do search for packages, and no one indicated that she/he never looks for npm packages. Our survey participants also reported that they mainly use web search engines (e.g., Google) when they search for npm packages to use. Interestingly, only 20% of them indicated they use other search engines.

Overall, the background information about the developers who participated in our survey shows that they are experienced in JavaScript and selecting npm packages, which gives us confidence in the finding based on their experiences.

3.1.4. Analysis method

To analyze our survey responses about the different factors used to select npm packages, we first showed the distribution of the Likert-scale for each factor, which ranges from 1 = not important to 5 = very important (Oppenheim, 1992). Second, for all responses of each factor, we calculated values of the median, the interquartile range (IQR), the mean, and the standard deviation (SD).

In addition, to analyze the free-text answers from the open-ended question related to developers' opinions, we performed an iterative coding process to understand whether the responses indicated any other factors that we did not consider in our survey (Rea and Parker, 2014). The first two authors iteratively developed a set of codes based on an inductive analysis approach (Seaman, 1999). In total, the authors manually examined 30 responses from the developers who answered the optional open-ended question. However, based on this analysis, we did not find any new factors that we did not consider in our survey. In fact, all the responses to this open-ended question supported the developers' opinions about the studied factors.

3.2. Study results

Table 3 shows the factors' name and the 5-point Likert-scale distribution for each factor from our survey responses. The table also shows the scale's median alongside the value of IQR and mean alongside SD. Overall, based on our survey results, we can divide the factors used by developers when selecting packages into three groups: (1) Mostly important factors (e.g., documentation, downloads, and stars), (2) somewhat important factors (e.g., license and testing), and (3) mostly unimportant factors (e.g., watchers and badges). In the following, we discuss the developers' perceptions in more detail:

Mostly important factors: On a 5-point scale, participants indicated that the most important factor when looking for an npm package to select is how well a package is documented. Table 3 shows that the majority 93% (median = 5.00 and mean = 4.65) of the responses agree with the statement that the GitHub repository of a npm package that they are examining to select should have some form of documentation. In addition, to confirm this statement, developer P40 stated that *"Sample code documentation on its usage"* are important factors when selecting an npm package to use.

The second most important factor reported by our survey participants is the number of downloads that the packages have. More than 85% (median = 4.5 and mean = 4.30 on the 5-point scale) of the responses said that they considered the number of downloads a package has when searching for a package to use. These results give a high indication that developers still consider the download count of packages as a sign of the community interest, which means that the package is a good option to select.

Table 3
Survey results of the factors used in selecting a package from the npm ecosystem.

Factor	Distribution 1 2 3 4 5	Median	IQR	Mean	SD
Documentation		5.0	0.0	4.64	0.77
Downloads		4.5	1.0	4.30	0.88
Stars		4.0	2.0	3.97	1.13
Vulnerabilities		4.0	2.0	3.70	1.24
Release		4.0	1.0	3.48	1.10
Commit Frequency		3.0	1.0	3.33	1.23
Closed Issues		3.0	1.0	3.29	1.09
License		3.0	3.0	3.26	1.39
Usage		3.0	2.0	3.19	1.33
Test Code		3.0	2.0	3.14	1.31
Dependencies		3.0	2.0	3.12	1.33
Contributors		3.0	2.0	3.03	1.40
Build Status		3.0	2.0	2.89	1.31
Website		3.0	3.0	2.67	1.32
Watchers		3.0	3.0	2.53	1.25
Badges		2.0	2.0	2.25	1.20
Forks		2.0	2.0	2.12	1.25

Our survey showed that developers also look for the number of stars the packages have when searching for a package to select. On the 5-points scale, developers believed that the reputation of the packages in terms of stars count is an important indicator with median = 4.0 and mean = 3.97. For example, developer P74 stated that *"reputation/popularity"* are the most important factors when selecting an npm package to use.

The fourth most important factor developers consider when searching for a new npm package to use is that the packages do not depend on vulnerable code. On a 5-point scale, 62% of the developers saw vulnerabilities as an essential factor when finding and selecting relevant npm packages. Furthermore, some developers in our survey emphasized the essentiality of this factor, participant P58 said *"... not dependent on other out of date or vulnerable packages"*. Also, participant P45 stated that they look for packages that are free of vulnerable code and the maintainers of the packages use tools to scan for vulnerabilities such as Snyk and dependabot tools.

Somewhat important factors: Our survey also revealed that there are some other factors that developers did not have an agreement on whether they are essential when they search for packages to select or not. We found that factors such as release (median = 4.0 and mean = 3.48), commits frequency (median = 3.0 and mean = 3.33), and test code (median = 3.0 and mean = 3.14) do not have a consistent agreement amongst the participants in our survey. However, some participants explicitly highlighted the importance of some of these factors, such as developer P69, who said *"examining the package repository and see the recent and historical activity/commits/updates would help making the decision"*. Another developer, P1 explained that *"...the test coverage status is useful, but can be verified manually in the code when deciding to use the package. The last date a commit was made is very important. The more recent the better. The last date a release was made is very important. The more recent the better"*. In addition, we observed from Table 3 that developers in our survey did not have a consistent agreement about factors such as license and number of dependent applications, number of dependencies that the package uses, the number of closed issues, and the number of contributors, which have, on a 5-point scale, values with a mean of 3.26, 3.19, 3.12, and 3.03, respectively.

Mostly unimportant factors: The other interesting group of the studied factors that developers tend not to consider when examining an npm package to use are: forks, badges, watchers, website, and build status. Our analysis showed that these factors received median values between 2.0 and 3.0 and mean values between 2.12 and 2.89 on a 5-point scale. However, only one developer from our survey supported the idea that examining the build status is essential when selecting a package to use and said P1 *"The build status is important no matter if it comes from Travis CI or other providers ..."*.

Finally, we observed that developers mentioned a few other factors when looking for npm packages. Our survey participants indicated that if there is a big software company that supports the package. For example, developers P39 said *"The source of the package, if it is by a company that actively supports open source and maintains their open source packages (ex: Facebook, Formidable labs, Infinite Red), brings more points"*. Also, another participant stated the same, P11 *"Private support for big companies in open source projects or libs (angular-google, react-facebook, etc.) that means the package usually follow good practices, test, linter, ci, etc, and the team that maintains the package is really good"*.

In addition, two other developers in our survey indicated that support of community discussions about the packages matters. For example, P94 mentions *"Whether the package is actively maintained by developers well known and reputed in the community & whether the package has good typescript support"* and P60 said *"If the library is supported with an online community where usage is discussed"*. Another developer, P20 stated *"References on other professional webpages about the package"*.

> In summary, JavaScript developers have access to a wealth of information about a large number of npm packages that can be used when deciding which packages to select. Our survey shows that developers mainly consider packages that are well-documented, popular, and do not suffer from security vulnerabilities. Moreover, when we conducted our survey, among the 118 respondents, 73 (62%) provided their emails and showed interest in our findings. This indicates the strong relevance and importance of the findings to the practitioners and the overall JavaScript development community.

4. Quantitative analysis

The goal of this analysis is to triangulate our qualitative findings. In particular, we wanted to quantitatively validate the developers' perception about the factors that highly-selected npm packages possess. In this analysis, we examined 2592 npm packages divided into highly-selected and not highly-selected packages. For each package in our dataset, we collected quantitative data to present the factors studied in our survey. Then, we used regression analysis to quantitatively investigate which of the studied factors are the most important.

4.1. Study design

In this section, we described our methodology of collecting a dataset of highly-selected and not highly-selected npm packages. We also described how we collected the studied factors, which served as the dependent variables in our study. Finally, we presented our analysis method and steps.

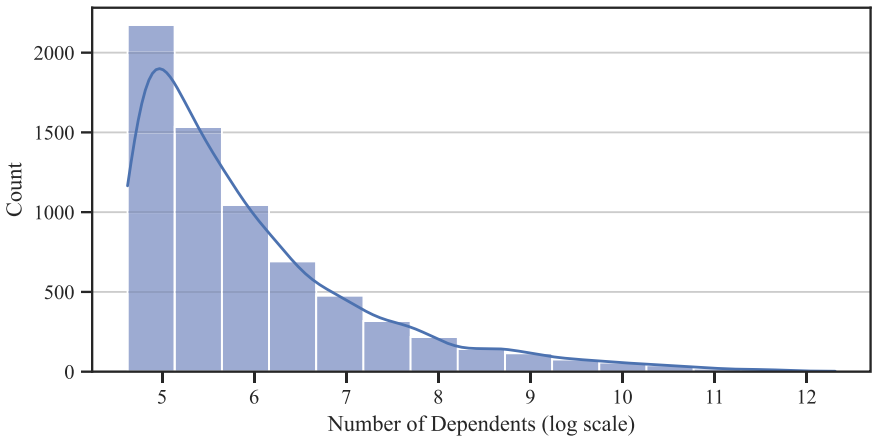

Fig. 4. A histogram for the used npm packages.

4.1.1. Data collection

To quantitatively examine the factors that make some npm packages highly-selected, we wanted to have a sufficient number of packages that present both highly-selected and not highly-selected packages. To do so, we resorted to study packages from the npm ecosystem. We started by retrieving the metadata information of all the npm packages that are published on the npm ecosystem. In particular, we wrote a crawler to interact with the npm registry and download the package.json file of every npm package (npm, 2017). It is important to note that the package.json contains all the package information, including the names of other packages that the package depends on them. Once we have the package.json, we started recursively analyzing the package.json file of every package to extract its dependencies. After that, for each package in the npm ecosystem, we counted the number of other packages that are listed as dependencies, i.e., the number of dependent packages.

It is important to note that we chose to use the number of dependent packages as a proxy of highly-selected packages from the npm ecosystem over other measurements, particularly download count, for two main reasons. First, the npm provides only an accumulated download count over time, which does not show the current stats of the package. Second, the download count that npm provides could include crawlers and downloads due to transitive dependencies. In addition, we used the number of dependent packages to distinguish between highly-selected and not highly-selected packages to quantitatively examine the studied factors' impact without being biased by the developers' opinions in the qualitative analysis. Furthermore, in our process, we considered only the direct dependent packages from the npm registry and avoided including dependent applications from open-source hosting services like GitHub. We did this since our goal is to proxy how many times a developer went through the process of selecting a package and decided to select the subject package. Moreover, platforms like GitHub hosts millions of applications created using predefined project templates or bootstrapping tools. For example, the tool *Create React App* alone bootstrapped millions public applications on GitHub, which may not be completed. Considering such applications from GitHub will amplify the decision taken by the creators of such tools or templates to overtake the decisions of millions of developers. In addition, we did not consider the number of transitive dependents because it does not reflect how many times developers have selected a package.

In total, we analyzed the package.json file of 1,423,956 npm packages. After that, we chose to study 6924 packages that have more than 100 dependent packages, i.e., the number of packages that depend on the selected packages. We decided to study npm packages that have more than 100 dependent packages for two main reasons. First, we found that prior work indicated that npm ecosystem has many packages that are not used, e.g., toy packages (e.g., Zerouali et al., 2019b; Abdalkareem et al., 2017). Thus, selecting packages with more than 100 direct dependent packages eliminated incompetent packages. Second, since we wanted to examine highly-selected npm packages, we focused on packages that can potentially be used and appear as an option for developers when searching for an npm package to use, for example, packages that are widely adopted by other packages. Moreover, we selected this threshold after examining the distribution of the number of dependent packages across all packages in the npm ecosystem.

Next, we sorted the selected npm packages based on their number of dependent packages. Fig. 4 presents the distribution of the number of dependent packages. We considered the top 20% based on the number of dependent packages as highly-selected packages and the bottom 20% as not highly-selected packages. We resorted to using these thresholds to have an essential distinction between the two samples and eliminate the gray area between them. Also, prior studies used a similar sampling technique (Bavota et al., 2015; Lee et al., 2020; Tian et al., 2015). In the end, we had 1385 highly-selected packages and 1385 not highly-selected packages. We used these packages in our quantitative analysis.

4.1.2. Package usage factors

Since we wanted to use regression analysis to understand the most important factors in determining highly-selected packages, we collected seventeen package factors. These factors are based on the ones we studied in the qualitative analysis. Since these factors present information that developers can observe by examining online sources about the npm packages, we resorted to extracting these factors from four different sources: (1) GitHub, which presents the package's source code and other development activities such as issues and commits, (2) npm, which contains information about npm packages that developers can examine on the npm website, (3) *npms*, which is the official search engine used by the npm platform and provides metadata about the packages, and (4) Snyk, which is a service that provides a dataset of vulnerable npm packages and their versions. Table 4 shows the factors with their names, value types, and descriptions. In the following, we presented the detailed process of extracting the studied factors from each data source:

GitHub: to collect the repository level factors, we used the official GraphQL API (GitHub, 2021a) to collect the number of forks, watchers, stars, and closed issues for each npm package in our dataset. Since GraphQL API does not provide direct access

Table 4
List of factors values with their description.

Factor	Type	Description
Forks	Number	Forks count on GitHub
Watchers	Number	Watcher count on GitHub
Contributors	Number	Contributors count on GitHub
Downloads	Number	Downloads count from npm
Stars	Number	Stars count on GitHub
Dependencies	Number	Count of dependencies from package.json
License	Boolean	Whether has a permissive license
Documentation	Number	Size of README file
Test code	Boolean	Whether has a test script
Build status	Number	Percentage of failed jobs on last commit
Vulnerabilities	Number	Percentage of vulnerable versions
Badges	Number	Count of badges in the README file
Website	Boolean	Whether has a website
Releases	Number	Frequency of releases
Closed issues	Number	Count of closed issues on GitHub
Commit frequency	Number	Count of commits in the last year
Usage	Number	Count of dependent repositories on GitHub

to the number of contributors and build status of each package repository, we used the GitHub REST API (GitHub, 2021b) to count the number of contributors and the list of build status. In addition, to measure the commits frequency, we cloned the GitHub repositories for each of the studied packages and counted the number of commits on all branches that were committed in the latest year. Finally, we wrote a web crawler to collect the package usage factor from the GitHub web interface, which presents the number of other GitHub repositories that depend on the package. Our crawler visited the GitHub repository page for each package and read the number of dependents from the dependency graph page on the GitHub website.

npm: from the official npm registry, we retrieved the list of releases for each package in our dataset. Then, we calculated the release frequency factor by dividing the number of releases by the number of days. Likewise, to present the dependencies factor, we used the registry to count the number of dependencies that a package uses in its last version. Also, to calculate the documentations factor for each package, we considered the size of the readme file. We then measured its size in terms of the number of its characters.

Also, we used both the npm registry and GitHub GraphQL API consecutively to retrieve the name of the license that a package declares. We then classified the licenses into three categories: (1) permissive licenses, (2) weak copyleft licenses, and (3) strong copyleft licenses (Meloca et al., 2018; Team, 2019). Finally, we retrieved the list of badges for each package using a tool called detect-readme-badges.[2] Once we had the list of badges, we calculated the badges factor by counting the number of badges used by the package.

***npms*:** for the download factor, we used the official npm search (*npms*.[3]) through its API to collect the number of downloads Next, we examined whether the package has a test code to represent the test code factor by querying the npms API. Additionally, we used the *npms* API to determine the website factor. To do so, we extracted the website URL for each package in our dataset. Since some packages refer to their GitHub repository as their main website, we filter out those URL addresses.

Snyk: to collect the vulnerabilities factor for each package in our dataset, we wrote a web crawler to collect the list of vulnerable releases from the Snyk web interface. Then, we divided the number of vulnerable releases by the total number of releases for each package to calculate the vulnerabilities factor.

[2] https://www.npmjs.com/package/detect-readme-badges.
[3] https://npms.io.

Table 4 shows the name, value type, and description of the factors that we used to build our logistic regression models. Since some packages do not have values for some factors, we filter out these packages from our dataset. In the end, we were able to collect factor values for 1332 highly-selected packages and 1195 not highly-selected packages.

4.2. Analysis method

To quantitatively examine the most impactful factors that determine highly-selected packages, we used logistic regression analysis. In our study, we examined the studied 2527 packages, which we classified into highly-selected and not highly-selected packages. We then built a logistic regression to model the dependent variable, whether a package is highly-selected or not highly-selected. In the following sections, we described the steps used to build the logistic regression model.

4.2.1. Correlation analysis

Since the interpretation of the logistic regression model can be affected by the highly correlated factors (Midi et al., 2010), we first started by removing highly correlated factors in our dataset. Thus, we computed the correlation among the independent variables using Spearman's rank correlation coefficient. We used Spearman correlation because it is resilient to non-normally distributed data, which is the case for our independent variables (Kendall, 1938). We considered any pair of independent variables that have a Spearman's coefficient of more than 0.8 to be highly correlated. We selected the cutoff of 0.8 Spearman since prior work suggested and used the same threshold for software engineering data (e.g., Tian et al., 2015; Li et al., 2017). Fig. 5 shows the hierarchical clustering based on the Spearman correlation among our independent variables. From Fig. 5, we observed that three factors are highly correlated, which are stars, forks, and watchers. Finally, for these three factors, we only kept the factor that is easy to interpret, which is the number of stars. After this analysis, we ended up having fifteen unique variables.

4.2.2. Redundancy analysis

Once we removed the highly correlated factors, we also applied redundancy analysis to detect variables that do not add information to the regression analysis (Harrell, 2015). Thus, we removed them so they do not affect the interpretation of our logistic regression model. In our dataset, we did not find redundant variables among the remaining fifteen factors.

4.2.3. Logistic regression

To build our logistic regression model, we followed steps that have been applied in prior studies (e.g., Lee et al., 2020). After identifying the factors that may impact the selection of an npm package, we used logistic regression to model the highly-selected packages. Since prior studies showed that using logistic regression may be affected by the estimated regression coefficient (Harrell, 2015; Lee et al., 2020), we trained our model using several bootstrap iterations. Similar to prior work (e.g., Lee et al., 2020), we created 100 rounds of bootstrap samples with a replacement for training and testing sets that ensure the testing samples were not included in the training set and vice versa. Then, we built a logistic regression model on the created bootstrap training samples, one for each iteration (i.e., 100 times) and test it on the testing samples. In the end, we calculated the mean of the sample statistics out of the 100 bootstrap samples.

4.2.4. Evaluating performance

Once we built our logistic regression model, we next wanted to examine the performance of the built model. Hence, we used

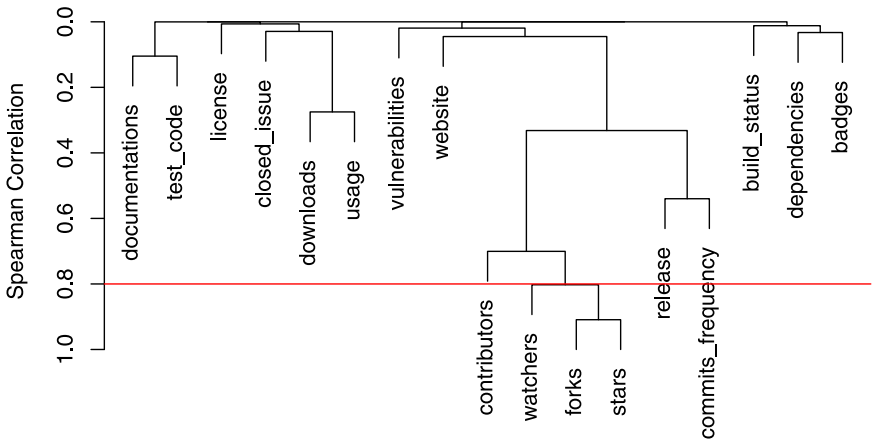

Fig. 5. The hierarchical clustering shows the factors that might impact the selection of npm packages. We apply the Spearman correlation test and use a cut-off value of 0.8, to eliminate highly correlated factors. This analysis left us with fifteen factors that is used in the regression analysis.

the area under the receiver operating characteristic curve (ROC-AUC), which is an evaluation measurement known for its statistical consistency (Bradley, 1997). An ROC-AUC value ranges between 0 and 1, where 1 indicates perfect prediction results, and 0 indicates completely wrong predictions. Accordingly, prior studies showed that achieving a 0.5 ROC-AUC value indicates that the model's predictions are as good as random. However, a ROC-AUC value equal to or more than 0.7 indicates an acceptable model performance for software engineering datasets (Nam and Kim, 2015; Lessmann et al., 2008; Yan et al., 2019). In our study, the logistic regression model achieved an ROC-AUC value of 0.74.

4.2.5. Factors importance

To investigate which of the examined factors are the most impactful in our logistic regression modeling of highly-selected packages, we used the Wald χ^2 maximum likelihood tests value of the independent factors in our model (Harrell, 2015). The higher the Wald χ^2 statistics value of an independent factor, the greater the probability that its impact is significant.

We also generated nomogram charts to present the studied factors' importance on our logistic regression model (Harrell, 2015; Iasonos et al., 2008). Nomograms are easily explainable charts that provide a way to explore the explanatory power of the studied factors. Since Wald χ^2 test provides us with only the explanatory power, we used the nomogram to show us the exact interpretation of how the variation in each factor influences the outcome of the regression model. Furthermore, the Wald χ^2 does not indicate whether the studied factors have positive or negative roles in determining highly-selected packages or not, while the nomogram provides such information.

Fig. 6 shows the generated nomogram of the logistic regression model. The line against each factor in the figure presents the range of values for that factor. We used the points line at the top of the figure to measure the volume of each factor contribution, while the total points line at the bottom of the figure presents the total points generated by all the factors. In our analysis, the higher the number of points assigned to a factor on the *x*-axis (e.g., the number of stars has 100 points), the larger its impact is on the logistic regression model. Having a higher value on the total points line reflects that a package will be more likely to be selected.

4.3. Study results

Table 5 shows the values of the Wald χ^2 and the *p*-value for the selected fifteen factors that may impact the highly-selected npm packages. Fig. 6 also shows the estimated effect of our factors using nomogram analysis (Iasonos et al., 2008). Overall, we observed that the regression analysis complemented the main qualitative findings. However, it controverted with the importance of some factors.

Table 5
The result of our logistic regression analysis for investigating the most important factors.

Factors	Wald χ^2	*p*-value	
Downloads	63.00	0.000	***
Stars	24.21	0.000	***
Closed issue	17.62	0.000	***
Vulnerabilities	16.47	0.000	***
Badges	12.21	0.001	***
Documentation	11.61	0.001	***
Dependencies	8.04	0.005	**
Build status	5.54	0.019	*
Test code	4.62	0.032	*
Contributors	4.41	0.036	*
Commits frequency	2.34	0.126	
Release	2.32	0.127	
License	1.68	0.196	
Usage	1.15	0.283	
Website	0.02	0.880	

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1.

From Table 5, we observed that the number of downloads has the most explanatory power with a Wald χ^2 value equal to 63.00 when we modeled the probability of highly-selected npm packages. The second most important factor in modeling highly-selected packages is the number of stars a package has (Wald $\chi^2 = 24.21$). Fig. 6 also shows that npm packages that have a high number of downloads and received a high number of stars have a high chance to be highly-selected packages.

Our regression analysis also showed that documentation and vulnerability factors have explanatory power as well. Interestingly, developers reported these two factors in our survey to have a high impact when selecting npm packages. With a Wald χ^2 value equal to 16.47, packages that have a high percentage of vulnerable versions have higher impact power and the same apply for the size of the readme files with Wald $\chi^2 = 11.61$. In addition, Fig. 6 confirms that documentation and vulnerabilities have a positive contribution to the probability of a npm package being highly-selected.

Interestingly, our regression analysis showed two of the studied factors that have an explanatory power when they are used to model the probability of highly-selected npm packages, which

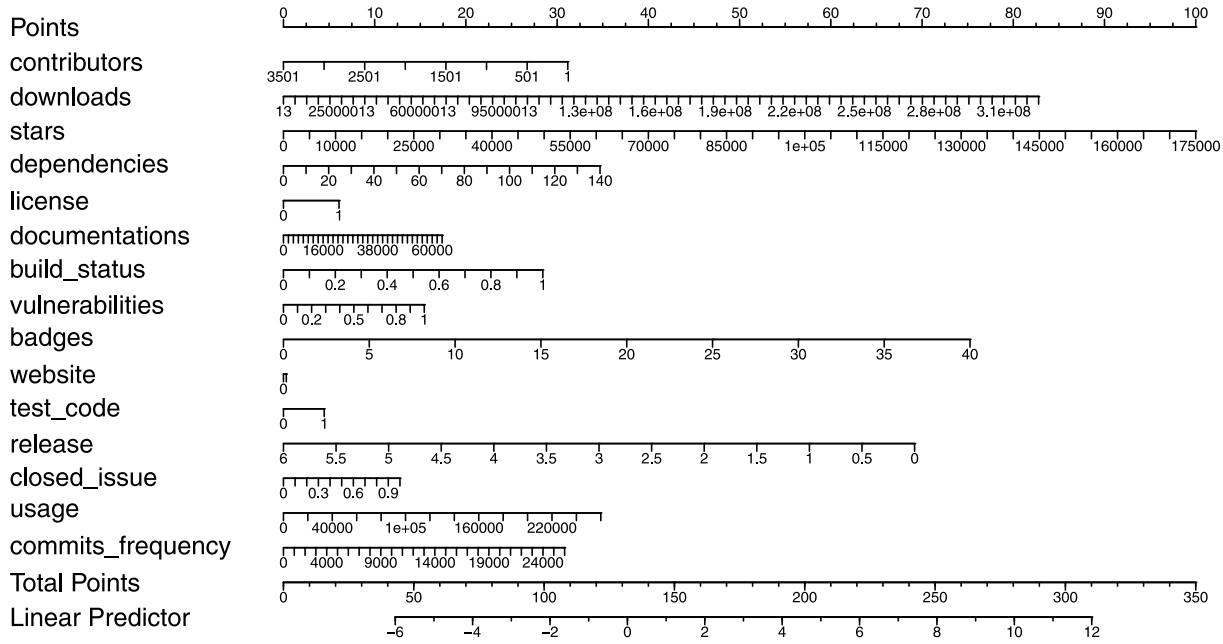

Fig. 6. The nomogram visually presents the impact of each of the studied factors in determining highly-selected npm packages. The logistic regression model used to generate this nomogram achieved a median ROC-AUC of 0.74 on 100 out-of-sample bootstrap iterations.

are the number of badges that the package has and the number of closed issues on Github. However, our survey results showed that developers tend not to consider these factors when searching for an npm package to use. Table 5 shows that the number of closed issues is the third most important factor with a Wald χ^2 value equal to 17.62 while the number of badges is placed fifth, having a value of Wald χ^2 equal to 12.21. Furthermore, Fig. 6 shows the number of badges has a positive contribution with the probability that the package will be highly-selected packages.

In addition, our nomogram analysis shows that the number of contributors as a factor has a negative contribution to the probability of a package being highly-selected. In contrast, the regression analysis showed that this factor has a modest explainable power (Wald $\chi^2 = 4.41$).

Overall, our analysis shows that there is a statistically significant difference between the factors of highly-selected and not highly-selected packages. Therefore, the question naturally arises here: do highly-selected packages provide functionalities different from not highly-selected packages? In particular, we are interested in understanding if these highly-selected packages provide functionalities that not highly-selected packages lack (e.g., packages providing functionalities related to working with arrays or strings). To this aim, we extracted the list of keywords provided by the publishers of the packages in our dataset to describe the key functionalities that the packages provide. Then, for the dataset of highly-selected and not highly-selected packages, we count the frequency of the keywords and present them as word clouds.

Fig. 7 reports the frequency of the keywords that developers wrote to describe their packages. In general, we observe that both word clouds contains similar keywords. For example, we see that highly-selected and not highly-selected packages provide test-related functionalities and CSS related functionalities. These analyses indicate that in high level, highly-selected and not highly-selected packages tend to perform functionalities from the same domains.

> In summary, our quantitative analysis complemented developers' perceptions about the factors that they look for when selecting an npm package to use. In particular, our results showed that highly-selected npm packages tend to possess characteristics that include a high number of downloads, stars, and a higher ratio of closed issues. Lastly, in contrast to our qualitative analysis results, our regression analysis showed that a higher number of badges is an essential characteristic of highly-selected npm packages.

5. Discussion

Our study has many direct benefits for the ecosystem maintainers and the npm community, particularly package owners and developers who use the npm packages. We discussed these implications and benefits in the following.

The npm software ecosystem maintainers should pay attention to certain aspects of the packages when building package search or recommendation tools. Several package search tools have been proposed and deployed, which can be classified into two main categories. The first category based on keyword search (npm-Documentation, 2017; Temple, 2017; Kashcha, 2017). These tools are limited since they do not take into consideration the quality aspect of the packages. Tools from the second category provide package search while considering some quality aspects of the packages, e.g., the *npms* tool (Cruz and Duarte, 2018). While *npms* is the official search tool used by the official npm website, it has some limitations. The main limitation of *npms* is that it assigns different weights of the used aspects without a clear justification, which negatively affects the quality of the search engine (Abdellatif et al., 2020). Our examination of *npms*' source code and documentation shows that *npms* arbitrarily gives weights to certain aspects when ranking the packages. We recommend that the npm ecosystem could use our results to build

(a) Highly-selected npm Packages. (b) Not highly-selected npm Packages.

Fig. 7. Word cloud of keywords description for highly-selected packages and not highly-selected packages.

more robust search tools. For example, the npm maintainers can integrate our rank of the important factors to weigh each factor's contribution when used in a search tool, which they are based on developers' perceptions.

Several package characteristics should be carefully examined by developers when choosing an npm package to depend on in their projects. As mentioned earlier, our results indicated that highly-selected packages possess specific characteristics. For example, our regression analysis results showed that the number of closed issues in the package repository is commonly related to highly-selected packages. We believe that JavaScript developers can use our results to build systematic guidelines for choosing an npm package to use. In fact, there have been several attempts to help developers create such a guideline (Franch and Carvallo, 2003; Wasike, 2010; Semeteys, 2008). However, their main drawback is that they focused on selecting packages in a specific context or propose general guidelines to select open source components. In addition, they do not consider package characteristics that npm provide, such as the number of downloads.

To promote their packages, the owner of npm packages should provide clear indications of their packages' characteristics. Gaining more popularity within the software ecosystem requires putting more effort into signaling the published packages' quality. Overall, all the package factors that our qualitative and quantitative results highlighted are essential factors that package owners can employ to attract more users. For example, many responses indicated that package documentation is an important factor when looking for a package to use. Based on these findings, we recommend developers invest more effort in making their package documentation, particularly readme files, clearer and up to date.

Package selection factors should be measured in the context of what they reflect. Developers could use some factors as a proxy to reflect other aspects of the projects, e.g. checking the commit frequency as an indication of project activity. So, measuring a factor in isolation from other factors that indicates the same aspects could be misleading. We can see in Table 6 that Commit Frequency has a lower rank in the quantitative results than the qualitative results. The same applies to Releases, another factor related to project activity. At the same time, we see that the Closed Issues factor has a higher rank in the quantitative results than the qualitative results, even though it reflects the same aspect, project activity.

6. Threats to validity

In this section, we discuss the potential threats to the validity of our work.

Table 6
Qualitative results vs. quantitative results.

Factor	Rank		Shift
	Qualitative	Quantitative	
Documentation	1	6	↓ 5
Downloads	2	1	↑ 1
Stars	3	2	↑ 1
Vulnerabilities	4	4	−−
Release	5	12	↓ 7
Commits frequency	6	11	↓ 5
Closed issue	7	3	↑ 4
License	8	13	↓ 5
Usage	9	14	↓ 5
Test code	10	9	↑ 1
Dependencies	11	7	↑ 4
Contributors	12	10	↑ 2
Build status	13	8	↑ 5
Website	14	15	↓ 1
Badges	15	5	↑ 10

6.1. Internal validity

Internal validity concerns factors that could have influenced our results. To qualitatively understand the factors that may impact the use of an npm package, we surveyed JavaScript developers. While we carefully designed our survey based on the guideline provided in Dillman (2011), our survey might have been influenced by some factors. First, our survey participants might poorly understand some of the factor statements. To mitigate this limitation, we conducted a pilot survey where we gave our survey to three expert JavaScript developers and incorporated their feedback about the survey. Second, we had a list of well-defined factors that may impact selecting an npm package. Even though we choose to study these factors since they are used in the literature and can be easily examined by developers, we may miss some other factors. To mitigate this threat, in our survey, we had one open-ended question, where we asked developers to provide us with any factors that are missed in our survey (Dillman, 2011). That said, none of our survey responses reported any new factors that can be quantitative.

To recruit participants in our survey, we resorted to developers who publish and use packages from the npm ecosystem. At the beginning of the survey, we articulated that the purpose of our study is to understand how developers select npm packages. This description may attract more attention from developers, who use npm packages more.

6.2. Construct validity

Construct validity considers the relationship between theory and observation in case the measured variables do not measure

the actual factors. In our study on the npm ecosystem, we used *npms* platform (Cruz and Duarte, 2018) to measure various quantitative factors related to download counts, testing, and having a website. Our measurements are only as accurate as *npms*; however, given that *npms* is the main search tool for npm, we are confident in the *npms* metrics. We also used Snyk (2021) to calculate the number of vulnerabilities that affect the studied packages. Thus, our analyses are as accurate as Snyk dataset. That said, we resorted to using the Snyk data since it has been used by other prior work (e.g., Alfadel et al., 2021; Zapata et al., 2018; Decan et al., 2018; Chinthanet et al., 2021). In addition, we wrote a crawler to extract factors from the Github platform through the use of Github API, so our collected data might be affected by the accuracy of this public API. Furthermore, In our study, we investigated package factors that can be observed in a mechanical way (e.g., examine the Github repository of the package). However, developers might select npm packages based on a discussion or recommendation by other developers. Thus, our studied factors may not present the whole picture. Further, to ensure that our survey participants have sufficient experience in JavaScript development and in selecting and using npm packages, thus, we asked the participants three questions related to their development experiences in our survey. The participants are asked to choose one of the following options (scale): <1 year, 1–3, 4–5, or more than five years for each question. As a result, those questions may be limited to presenting all participants with more than five years of experience as one group.

6.3. External validity

Threats to external validity concern the generalization of our findings. In our study, we investigated the factor that impacts highly-selected packages that are published on the npm ecosystem. Our results might not be generalized to other software ecosystems, such as maven for Java or PyPi for Python. However, since npm ecosystem is the most popular software ecosystem, this gave us confidence in our results. Also, scientific literature showed that studying individual cases has significantly increased our knowledge in areas such as economics, social sciences, and software engineering (Flyvbjerg, 2006). Second, our dataset that was used in the quantitative analysis presents only open-source packages hosted on GitHub that do not reflect proprietary packages or packages that are hosted on other platforms such as GitLab and BitBucket. Furthermore, we surveyed 118 JavaScript developers, so we do not claim that our results are generalized to other developers who do not know JavaScript or the npm software ecosystem.

Finally, one criticism of empirical studies results is "I know it all along" thought or nothing new is learned. However, such common knowledge has rarely been shown to be trusted and is often quoted without scientific and research evidence. Our paper provides such evidence and supports common knowledge (e.g., "packages with good documentations tend to be highly-selected packages") while some are challenged (e.g., "developers do not consider the number of badges when selecting a new package to use".).

7. Conclusion

In this work, we used a mixed qualitative and quantitative approach to investigate the characteristic of highly-selected npm packages. We started by identifying seventeen packages selection factors based on our literature review and used by existing online package search tools. Then, we qualitatively investigated the factors developers look for when choosing an npm package by surveying 118 JavaScript developers. Second, we quantitatively examined these factors by building a logistic regression model using a dataset of 2527 npm packages divided into highly-selected and not highly-selected packages.

Among our main findings, we highlighted that JavaScript developers believe that highly-selected packages are well-document, receive a high number of stars on GitHub, have a large number of downloads, and do not suffer from security vulnerabilities. Moreover, our regression analysis complemented what developers believe about highly-selected packages and showed the divergences between the developers' perceptions and the characteristics of highly-selected packages.

CRediT authorship contribution statement

Rabe Abdalkareem: Supervision. **Emad Shihab:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank all the developers and managers who provided their valuable feedback during the survey.

References

Abdalkareem, R., 2017. Reasons and drawbacks of using trivial npm packages: the developers' perspective. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 1062–1064.

Abdalkareem, R., Nourry, O., Wehaibi, S., Mujahid, S., Shihab, E., 2017. Why do developers use trivial packages? An empirical case study on npm. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2017, Association for Computing Machinery, pp. 385–395.

Abdalkareem, R., Oda, V., Mujahid, S., Shihab, E., 2020. On the impact of using trivial packages: an empirical case study on npm and PyPI. Empir. Softw. Eng. 25 (2), 1573–7616.

Abdellatif, A., Zeng, Y., Elshafei, M., Shihab, E., Shang, W., 2020. Simplifying the search of npm packages. Inf. Softw. Technol. 126, 106365.

Alfadel, M., Costa, D.E., Shihab, E., 2021. Empirical analysis of security vulnerabilities in python packages. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 446–457.

Bavota, G., Linares-Vásquez, M., Bernal-Cárdenas, C.E., Penta, M.D., Oliveto, R., Poshyvanyk, D., 2015. The impact of API change- and fault-proneness on the user ratings of android apps. IEEE Trans. Softw. Eng. 41 (4), 384–407.

Begel, A., Bosch, J., Storey, M., 2013. Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. IEEE Softw. 30 (1), 52–66.

Borges, H., Valente, M.T., 2018. What's in a GitHub star? Understanding repository starring practices in a social coding platform. J. Syst. Softw. 146, 112–129.

Bradley, A.P., 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognit. 30 (7), 1145–1159.

Chatzidimitriou, K.C., Papamichail, M.D., Diamantopoulos, T., Oikonomou, N., Symeonidis, A.L., 2019. Npm packages as ingredients: A recipe-based approach. In: Proceedings of the 14th International Conference on Software Technologies - Volume 1: ICSOFT. INSTICC, SciTePress, pp. 544–551.

Chinthanet, B., Kula, R.G., McIntosh, S., Ishio, T., Ihara, A., Matsumoto, K., 2021. Lags in the release, adoption, and propagation of npm vulnerability fixes. Empir. Softw. Eng. 26 (3), 1–28.

Cruz, A., Duarte, A., 2018. npms. https://npms.io/about. (Accessed on 01/30/2021).

Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J., 2012. Social coding in GitHub: Transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. In: CSCW 2012, ACM, pp. 1277–1286.

de la Mora, F.L., Nadi, S., 2018. An empirical study of metric-based comparisons of software libraries. In: Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering. PROMISE '18, Association for Computing Machinery, New York, NY, USA, pp. 22–31.

DeBill, E., 2022. Modulecounts. http://www.modulecounts.com/. (Accessed on 01/25/2022).

Decan, A., Mens, T., Claes, M., 2017. An empirical comparison of dependency issues in OSS packaging ecosystems. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. SANER, pp. 2–12.

Decan, A., Mens, T., Constantinou, E., 2018. On the impact of security vulnerabilities in the npm package dependency network. In: Proceedings of the 15th International Conference on Mining Software Repositories. pp. 181–191.

del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., 2011. A survey on open source software trustworthiness. IEEE Softw. 28 (5), 67–75.

Dillman, D.A., 2011. Mail and Internet Surveys: The Tailored Design Method–2007 Update with New Internet, Visual, and Mixed-Mode Guide. John Wiley & Sons.

Flyvbjerg, B., 2006. Five misunderstandings about case-study research. Qual. Inq. 12 (2), 219–245.

Franch, X., Carvallo, J.P., 2003. Using quality models in software package selection. IEEE Softw. 20 (1), 34–41.

GitHub, 2021a. GitHub GraphQL API - GitHub docs. https://docs.github.com/en/graphql. (accessed on 01/27/2022).

GitHub, 2021b. GitHub REST API - GitHub docs. https://docs.github.com/en/rest. (accessed on 01/27/2022).

Gousios, G., Pinzger, M., Deursen, A.v., 2014. An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering. In: ICSE 2014, pp. 345–355.

Haefliger, S., von Krogh, G., Spaeth, S., 2008. Code reuse in open source software. Manage. Sci. 54 (1), 180–193.

Haenni, N., Lungu, M., Schwarz, N., Nierstrasz, O., 2013. Categorizing developer information needs in software ecosystems. In: Proceedings of the 2013 International Workshop on Ecosystem Architectures. In: WEA 2013, ACM, pp. 1–5.

Harrell, Jr., F.E., 2015. Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis. Springer.

Hata, H., Todo, T., Onoue, S., Matsumoto, K., 2015. Characteristics of sustainable OSS projects: A theoretical and empirical study. In: 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE, pp. 15–21.

Hauge, O., Osterlie, T., Sorensen, C., Gerea, M., 2009. An empirical study on selection of open source software - preliminary results. In: 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. IEEE, pp. 42–47.

Iasonos, A., Schrag, D., Raj, G.V., Panageas, K.S., 2008. How to build and interpret a nomogram for cancer prognosis. J. Clin. Oncol. 26 (8), 1364–1370.

Jadhav, A.S., Sonar, R.M., 2009. Evaluating and selecting software packages: A review. Inf. Softw. Technol. 51 (3), 555–563.

John, W., Creswell, P.C., CLARK, V., 2000. Designing and Conducting Mixed Methods Research. SAG.

Kashcha, A., 2017. npm packages sorted by pagerank. http://anvaka.github.io/npmrank/online/. (Accessed on 11/24/2017).

Kendall, M.G., 1938. A new measure of rank correlation. Biometrika 30 (1/2), 81–93.

Larios Vargas, E., Aniche, M., Treude, C., Bruntink, M., Gousios, G., 2020. Selecting third-party libraries: The practitioners' perspective. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 245–256.

Lee, D., Rajbahadur, G.K., Lin, D., Sayagh, M., Bezemer, C.-P., Hassan, A.E., 2020. An empirical study of the characteristics of popular Minecraft mods. Empir. Softw. Eng. 25 (5), 3396–3429.

Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. Softw. Eng. 34 (4), 485–496.

Li, H., Shang, W., Zou, Y., Hassan, A.E., 2017. Towards just-in-time suggestions for log changes. Empir. Softw. Eng. 22 (4), 1831–1865.

Meloca, R., Pinto, G., Baiser, L., Mattos, M., Polato, I., Wiese, I.S., German, D.M., 2018. Understanding the usage, impact, and adoption of non-OSI approved licenses. In: Proceedings of the 15th International Conference on Mining Software Repositories. MSR '18, Association for Computing Machinery, pp. 270–280.

Midi, H., Sarkar, S.K., Rana, S., 2010. Collinearity diagnostics of binary logistic regression model. J. Interdiscip. Math. 13 (3), 253–267.

Mujahid, S., 2021. Effective Dependency Management for the JavaScript Software Ecosystem (Ph.D. thesis). Concordia University, Montreal, Quebec ,Canada.

Mujahid, S., Abdalkareem, R., Shihab, E., 2022. Dataset: What are the characteristics of highly-used packages? A case study on the npm ecosystem. http://dx.doi.org/10.5281/zenodo.6592024.

Mujahid, S., Costa, D.E., Abdalkareem, R., Shihab, E., Saied, M.A., Adams, B., 2021. Toward using package centrality trend to identify packages in decline. IEEE Trans. Eng. Manage. 1–15. http://dx.doi.org/10.1109/TEM.2021.3122012.

Nam, J., Kim, S., 2015. CLAMI: Defect prediction on unlabeled datasets (T). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 452–463.

npm, 2017. registry | npm Docs. https://docs.npmjs.com/cli/v6/using-npm/registry. (accessed on 12/24/2020).

npm-Documentation, 2017. npm. https://www.npmjs.com/. (Accessed on 11/24/2017).

Oppenheim, A.N., 1992. Questionnaire Design, Interviewing and Attitude Measurement. Pinter Publishers.

Pano, A., Graziotin, D., Abrahamsson, P., 2018. Factors and actors leading to the adoption of a JavaScript framework. Empir. Softw. Eng. 23 (6), 3503–3534.

Qiu, S., Kula, R.G., Inoue, K., 2018. Understanding popularity growth of packages in JavaScript package ecosystem. In: 2018 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering. BCD, IEEE, pp. 55–60.

Qiu, H.S., Li, Y.L., Padala, S., Sarma, A., Vasilescu, B., 2019. The signals that potential contributors look for when choosing open-source projects. In: ACM Conference on Computer-Supported Cooperative Work and Social Computing. In: CSCW, ACM.

Rea, L.M., Parker, R.A., 2014. Designing and Conducting Survey Research: A Comprehensive Guide. John Wiley & Sons.

Seaman, C.B., 1999. Qualitative methods in empirical studies of software engineering. IEEE Trans. Softw. Eng. 25 (4), 557–572.

Semeteys, R., 2008. Method for qualification and selection of open source software. Open Source Bus. Resour..

Serebrenik, A., Mens, T., 2015. Challenges in software ecosystems research. In: Proceedings of the 2015 European Conference on Software Architecture Workshops. ECSAW '15, Association for Computing Machinery, New York, NY, USA.

Sheoran, J., Blincoe, K., Kalliamvakou, E., Damian, D., Ell, J., 2014. Understanding "Watchers" on GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories. In: MSR 2014, ACM, pp. 336–339.

Smith, E., Loftin, R., Murphy-Hill, E., Bird, C., Zimmermann, T., 2013. Improving developer participation rates in surveys. In: 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering. CHASE, IEEE, pp. 89–92.

Snyk, 2021. Snyk | Developer security | Develop fast. Stay secure. https://snyk.io/. (accessed on 01/31/2022).

StackOverflow, 2017. node.js - How to find search/find npm packages - Stack Overflow. https://stackoverflow.com/questions/10568512/how-to-find-search-find-npm-packages. (Accessed on 11/24/2017).

Team, S.E., 2019. Top open source licenses and legal risk. https://www.synopsys.com/blogs/software-security/top-open-source-licenses/. (accessed on 12/20/2020).

Temple, C., 2017. npm Discover · see what everyone else is using. http://www.npmdiscover.com/. (Accessed on 11/24/2017).

Tian, Y., Nagappan, M., Lo, D., Hassan, A.E., 2015. What are the characteristics of high-rated apps? A case study on free Android Applications. In: 2015 IEEE International Conference on Software Maintenance and Evolution. ICSME, pp. 301–310.

Trockman, A., Zhou, S., Kästner, C., Vasilescu, B., 2018. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In: Proceedings of the 40th International Conference on Software Engineering. ACM, pp. 511–522.

Wasike, S.N., 2010. Selection process of open source software component.

Xu, B., An, L., Thung, F., Khomh, F., Lo, D., 2019. Why reinventing the wheels? An empirical study on library reuse and re-implementation. Empir. Softw. Eng..

Yamashita, K., Kamei, Y., McIntosh, S., Hassan, A.E., Ubayashi, N., 2016. Magnet or sticky? Measuring project characteristics from the perspective of developer attraction and retention. J. Inf. Process. 24 (2), 339–348.

Yan, M., Xia, X., Shihab, E., Lo, D., Yin, J., Yang, X., 2019. Automating change-level self-admitted technical debt determination. IEEE Trans. Softw. Eng. 45 (12), 1211–1229.

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

Zerouali, A., Cosentino, V., Mens, T., Robles, G., Gonzalez-Barahona, J.M., 2019a. On the impact of outdated and vulnerable Javascript packages in Docker images. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, pp. 619–623.

Zerouali, A., Mens, T., Robles, G., Gonzalez-Barahona, J.M., 2019b. On the diversity of software package popularity metrics: An empirical study of npm. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, In: SANER 2019, IEEE, pp. 589–593.

Zheng, W., Zhang, Q., Lyu, M., 2011. Cross-library API recommendation using web search engines. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. ACM, pp. 480–483.

Suhaib Mujahid is a senior software engineer at Mozilla. He obtained his Ph.D. in Software Engineering from Concordia University, Canada. His research interests include mining software repositories, defect prediction and avoidance, empirical software engineering, and machine learning. Mujahid received his master's in Software Engineering from Concordia University in 2017. His work has been published at premier venues such as ICSE, FSE, ICSME, and MSR and major journals such as TSE, EMSE, and TEM. You can find more about him at https://suhaib.ca.

Rabe Abdalkareem is a lecturer in the department of computer science in the Faculty of Science at Omar Al-Mukhtar University. Previously, he was an assistant professor in the School of Computer Science at Carleton University. He obtained his Ph.D. in Computer Science and Software Engineering from Concordia University, Canada. His research investigates how the adoption of crowd-sourced knowledge affects software development and maintenance. Abdalkareem received his masters in applied computer science from Concordia University. His work has been published at premier venues such as FSE, ICSME, and MSR and major journals such as TSE, IEEE Software, EMSE, and IST. More information can be found at https://rabeabdalkareem.github.io/

Emad Shihab is a professor in the Department of Computer Science and Software Engineering at Concordia University. He received his PhD from Queens University. Dr. Shihabs research interests are in Software Quality Assurance, Mining Software Repositories, Technical Debt, Mobile Applications and Software Architecture. He worked as a software research intern at Research In Motion in Waterloo, Ontario and Microsoft Research in Redmond, Washington. Dr. Shihab is a member of the IEEE and ACM. More information can be found at http://das.encs.concordia.ca.