

# Comparison of JavaScript package managers

Alexander Jacobs



<b>Author(s)</b> Alexander Jacobs	
<b>Degree programme</b> Business Information Technology	
<b>Report/thesis title</b> Comparison of JavaScript package managers	<b>Number of pages and appendix pages</b> 32 + 1
<p>Package managers have long been used in software development to manage third-party code libraries or as part of operating systems such as Linux to manage installed software. With the release of the first publicly available package manager for JavaScript, NPM in 2010, these tools have become a central part of modern JavaScript development.</p> <p>The objective of this thesis research was to compare and analyze the three most prominent package managers in use by JavaScript developers today and to observe the benefits, drawbacks and operating performance of each in an effort to determine if there is a clear favorite out of the three or if there are benefits to using each of them depending on certain circumstances.</p> <p>To conduct this research, a survey was conducted to gather data from developers on their opinions and experiences using the software. Secondly, I performed tests using each package manager on different operating systems to measure the comparative performance.</p> <p>The conclusions drawn from this research was that while you may get better performance using one or the other depending on your hardware and operating system, I would recommend the usage of NPM as your main package manager because it currently has the strongest security features. I would not recommend using PNPM at this time as it lacks important security features to detect compromised packages. While some areas of package managers could be improved (security, error message clarity) their use are nonetheless strongly recommended by the developers surveyed.</p>	
<b>Keywords</b> JavaScript, package manager, Node.js, NPM, Yarn, ECMAScript, PNPM, web development	

## Table of contents

1	Introduction .....	1
1.1	Background.....	1
1.2	Objectives .....	2
1.3	Target audience .....	3
1.4	Research scope .....	3
1.5	Research methods.....	3
2	Research background .....	4
2.1	ECMAScript and module support within JavaScript.....	4
2.2	Overview of modern JavaScript package manager functions .....	5
2.3	NPM.....	7
2.4	Yarn .....	7
2.5	PNPM .....	8
3	Survey of JavaScript developers .....	10
3.1	The given questions .....	10
3.2	Analysis of the gathered data.....	11
3.2.1	Questions 1-7: Developer background .....	11
3.2.2	Questions 8-11, 14: Experience with package managers .....	15
3.2.3	Questions 12-13: Error messages.....	17
3.2.4	Questions 15-17: Security .....	18
3.2.5	Questions 18-19: Additional thoughts.....	20
4	Hardware Test.....	21
4.1	Hardware used.....	21
4.1.1	Computer Specifications .....	22
4.1.2	Operating Systems tested .....	22
4.1.3	VirtualBox specifications .....	22
4.2	Test results .....	22
4.2.1	Windows 10 .....	22
4.2.2	MacOS.....	23
4.2.3	Ubuntu 19.04 .....	23
5	Discussion.....	24
5.1	Reflection on my learning during the thesis process .....	24
5.2	Awareness and importance of package managers .....	24
5.3	Performance .....	25
5.4	Potential areas of improvement.....	25
5.5	Security.....	26
5.6	Observations of the ranked questions .....	27
6	Conclusion .....	28

References .....	30
Appendices .....	33
Appendix 1. Table of figures.....	33

# 1 Introduction

## 1.1 Background

The JavaScript programming language was first released to the world in September 1995, developed internally by Brendan Eich for the Netscape Communications Corporation (Brendan Eich 2008) as a small scripting language for adding interactive elements to web pages used with the Netscape Navigator web browser. Since then, it has become a central pillar of the world wide web as the language evolved beyond its “toy” origins to power such large-scale apps such as Google Earth.

As the language has evolved, so has the complexity involved in developing apps with it. To help in speeding up development and reducing time spent “re-inventing the wheel” libraries such as jQuery gained in popularity, allowing developers to leverage work done by others in the open source community to strengthen their own projects.

To handle managing these external code libraries, developers in languages such as Java started to gravitate to software tools called “package managers”. Package managers are designed to remove the tedium from integrating external code libraries. Common tasks such as installing and updating packages can be reduced to just a simple console command (Debian 2017).

For example, to include a code library without using a package manager, you would need to manually search for it on the internet, download the JavaScript file and link it into the HTML page of the web app you are developing. With a package manager, you could simply run a command such as “npm install jquery” and it will be installed into your project folder and you can import it into your code.



```
<html>
  <head>
    <script src="www.cdn.org/jquery-2.8.9.js"></script>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Figure 1: Example of manually importing a code library into a web page without a package manager

The rise of JavaScript package managers was preceded by the Node.js framework, the initial version of which released on May 27<sup>th</sup>, 2009. (Node.js Github 2019) This framework allowed JavaScript to be run outside of the browser on the V8 JavaScript Engine, with Node.js specific Application Programming Interfaces (APIs) to interact with the operating system. This framework allowed for the development of such applications as command line programs or server-side frameworks such as Express.js.

The first package manager designed for JavaScript, Node Package Manager (NPM), was first released on January 13<sup>th</sup>, 2010 and bundled with the Node.js framework. This allowed for the convenient swapping of open-source packages among JavaScript developers and as of 2018 was the clear market leader with 60% of surveyed users using NPM (Node.js User Survey 2018)

## 1.2 Objectives

While NPM is the largest package manager by market share as of the time of writing, there are other package managers that have found audiences in the developer community. While developers may have plenty of subjective opinions on the superiority of their preferred tools, the objective of this thesis research is to analyse the available package managers individually and comparatively, to observe their benefits, drawbacks, potential security issues they can bring to a software project or if, in fact, there is any significant difference between them at all.

### **1.3 Target audience**

The target audience of this research will be software developers that mainly use JavaScript in their studies or working careers.

I anticipate that this audience would benefit the most from the results of this research as it would allow them to make decisions in choosing a package manager that might benefit them in their current or future careers as JavaScript developers.

Potentially, they would also be the group that would best use the research results to contribute to their favourite package managers in the way of new features or improvements to currently existing features.

### **1.4 Research scope**

While there exists a multitude of available package managers, the scope of this research will be limited to three of the available package managers. This will allow for a manageable work schedule and focused results that would be of clear interest to the target audience of this thesis research.

NPM and Yarn were chosen as they are currently the two dominant package managers by the size of their market share compared to the other available solutions (Node.js User Survey 2018)

PNPM was chosen not due to its market share, as like other package managers outside the top two, it holds a market share around 0.1% (Node.js User Survey 2018) but due to design differences that stand in stark difference to the previous two that were chosen.

### **1.5 Research methods**

This research will consist of several methods used to gauge the opinions of JavaScript developers in student and working life and the comparative performance, benefits and potential flaws of the three chosen package managers.

A survey will be designed and sent out to the target audience of this research, and the results will be analysed as part of this research to see what kind of experiences developers have with package managers and their opinions on the package managers in the research scope.

Secondly, to monitor the comparative performance of the three package managers, tests will be conducted on a series of hardware and operating systems to observe performance when a reasonably large open-source JavaScript project is cloned from a GitHub repository to a desktop computer and packages are installed under a range of circumstances.

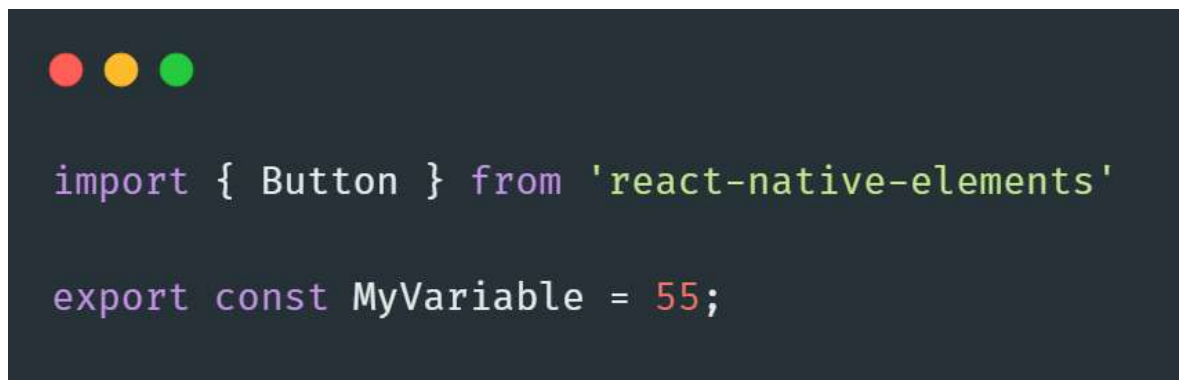
## 2 Research background

### 2.1 ECMAScript and module support within JavaScript

ECMAScript is the name given to the official standardization of the JavaScript language as standardised by ECMA International (MDN 2019) These standards aim to define functionality for the JavaScript language that will later be implemented into language interpreters for web browsers (interpreters read the code passed to them and executes it, in comparison to a compiler, which transforms the code into binary first and then runs it) since June of 1997, ECMA International have defined standards for new versions of JavaScript that implement ever increasing functionality.

The first three versions of ECMAScript were released on June 1997, June 1998 and December 1999. A 4<sup>th</sup> version was slated for completion on October 2008 but was abandoned due to political conflicts amongst the members of the standardising body (For example, Brendan Eich wrote a blog post accusing another member of the body of “spreading falsehoods” (Brendan Eich 2007))

Eventually, the first update to JavaScript in a decade was released as ECMAScript 5<sup>th</sup> edition on December 2009, a 5.1 version was released on June 2011 and the 6<sup>th</sup> edition was released on June 2015 which implemented native module support into JavaScript for the first time.



```
import { Button } from 'react-native-elements'  
  
export const MyVariable = 55;
```

Figure 2: An example of using the ECMAScript 6 module system for importing and exporting code



Before the 6<sup>th</sup> edition was released with native module support, Node.js implemented a method of importing and exporting modules, allowing for the easy implementation of third-party libraries into a JavaScript project (Node.js Docs 2019)



```
const express = require('express')

module.exports = {
  add: function(a, b) {
    return a + b;
  }
}
```

Figure 3: An example of using the Node.js module system for importing and exporting code

The ability to easily define and import / export packages allowed for the wide proliferation of open source packages for JavaScript, the growth of which was aided by NPM becoming an addition of the Node.js framework.

## 2.2 Overview of modern JavaScript package manager functions

In general, the purpose of a package manager is to handle the retrieval, installation, installation of dependent packages and the updating of these packages (or in the case of operating systems, managing external software)

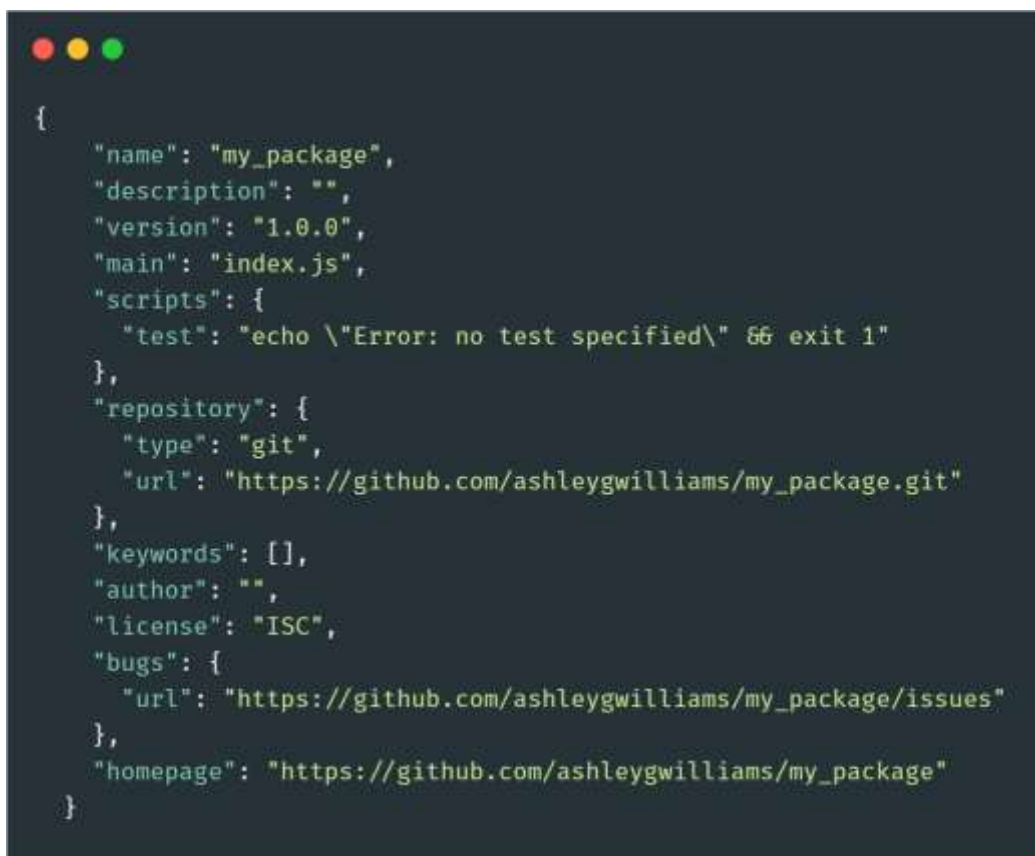
The first package managers developed during the early 90's for Linux operating systems were developed to solve the problem of having to download the source for programs and other dependant code needed for the program and having to manually compile it into an executable file that the end user could use. With package managers, the software could

search for the program the user specified, determine what other dependant programs would need to be installed and download a pre-compiled version onto the user's computer from a package repository service. Debian Linux was one of the first to include a package manager, with the DPKG tool being released in January 1994 (Debian 2015)

Package managers for programming languages essentially are designed to solve the same problem. Instead of a user having to manually search for a code library and any other dependant code they may need, the package manager does it for them.

All the tested package managers run on the Node.js framework, and generally handle the installation of packages the same way:

When a project is initialised using either of the package managers listed here, a "package" file is generated. Inside this file, it lists the name of the project, version number, the code repository, scripts used to run the program and a list of the dependencies (third-party code installed by the package manager for use with the project). All this information can be changed later within this file. (NPM Documentation 2019)

A screenshot of a code editor window showing the content of a package.json file. The code is written in JSON format and includes fields for name, description, version, main, scripts, repository, keywords, author, license, bugs, and homepage. The code is as follows:

```
{
  "name": "my_package",
  "description": "",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/ashleygwilliams/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/ashleygwilliams/my_package/issues"
  },
  "homepage": "https://github.com/ashleygwilliams/my_package"
}
```

Figure 4: Example of what a package file would look like when using NPM, taken from the NPM Documentation (NPM Documentation 2019)

When you run the install command for the first time, a second file is generated called the “lock” file. This file lists every package and dependant package that needs to be installed, and the specific version numbers of each file. This file can be quite long as many larger libraries or packages have a significant number of dependant packages. It is considered good practice to include the “lock” file inside your project repository so when others download and run the installation command, they test it with the same installed packages as you to ensure consistency. (NPM Documentation 2019)

While the names and format used for these files may differ, all the solutions tested use files like these to verify and install packages needed for a project. Currently all the package managers retrieve their files from the NPM package registry, so any library you search for will be able to be installed with all three solutions tested.

## **2.3 NPM**

Initially released on January 13<sup>th</sup>, 2010 (NPM Github 2019) and included as part of the Node.js framework, NPM is now the dominant package manager for JavaScript by market share, owning more than half of the userbase at 60% when last checked in 2018, helped in no small part to being bundled with Node.js. (Node.js User Survey 2018)

Currently the NPM package manager is maintained by NPM Inc, a company founded in 2014 for the purpose of maintaining the NPM software and package repository, while offering paid services and support for enterprise use (NPM Inc. 2019)

Development is currently very active, with the latest stable version (6.9.0) being released on March 6<sup>th</sup>, 2019 (NPM Github 2019)

The NPM registry, which is the service that hosts packages submitted by developers, is the single largest package registry online as of 2017, serving more than double that of the next largest registry, the Apache Maven registry for Java packages. (Linux.com 2017)

## **2.4 Yarn**

Yarn was initially released on June 17<sup>th</sup>, 2016 (Yarn Github 2019), developed by a team at Facebook to solve the problems of consistency, performance and security that they encountered as the size of their codebase grew significantly. (Code 2019)

Yarn claims to solve these problems through several design decisions: by utilizing a package cache to increase install times, an offline mode so previously installed packages can

be re-installed without an internet connection and a more efficient utilization of network requests to improve download speeds (Yarn Homepage 2019)

Yarn is the second-most popular package manager by market share, having 13% when last surveyed. Along with NPM, Yarn is one of the only JavaScript package managers to have a market share higher than 0.1% (Node.js User Survey 2018)

The Yarn package manager uses the NPM registry by default, allowing you to install all packages that you could with NPM, and the CLI (command line interface) utilizes similar commands as NPM, allowing the user of one to easily switch to the other.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal displays three sections of commands, each starting with a comment line. The first section shows installing a package locally with both npm and yarn. The second section shows installing a package globally with both npm and yarn. The third section shows uninstalling a package with both npm and yarn.

```
// Installing a package
> npm install react-devtools
> yarn install react-devtools

// Installing a package globally
> npm install -g expo-cli
> yarn install -g expo-cli

// Uninstalling a package
> npm uninstall react-devtools
> yarn remove react-devtools
```

Figure 5: An example of using different commands with NPM and Yarn

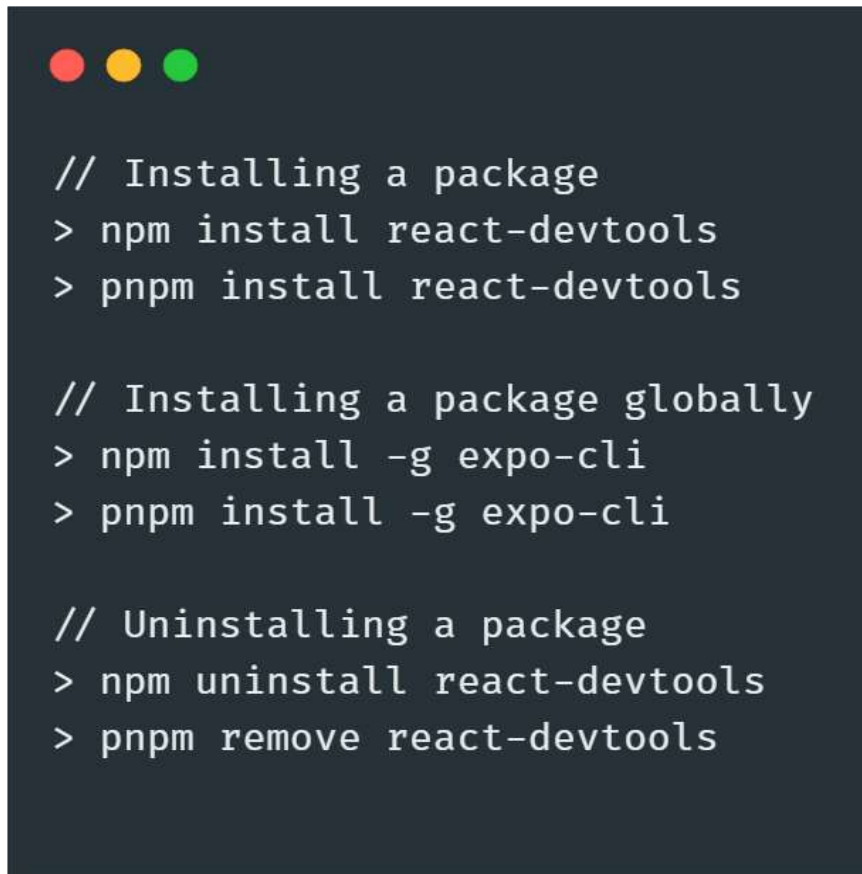
This also means that if a package is deleted from the NPM registry or is compromised in some way, this can affect users of Yarn as well which is a topic that will also be covered in this thesis research.

## 2.5 PNPM

The initial version of PNPM was released August 16<sup>th</sup> 2016 (PNPM Github 2019) to solve the problem of large “node\_modules” folder sizes. To do this, it ensures that all packages are only installed once into a central folder and any projects using those packages will

contain links to those packages, ensuring that no duplicate packages are installed. (PNPM Homepage 2019)

As with Yarn, the PNPM CLI aims to be as close to the NPM CLI as possible, allowing for an easy transition for NPM users.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal displays three sections of commands, each starting with a comment line. The first section shows installing 'react-devtools' locally with both 'npm' and 'pnpm'. The second section shows installing 'expo-cli' globally with both 'npm' and 'pnpm'. The third section shows uninstalling 'react-devtools' with both 'npm' and 'pnpm'.

```
// Installing a package
> npm install react-devtools
> pnpm install react-devtools

// Installing a package globally
> npm install -g expo-cli
> pnpm install -g expo-cli

// Uninstalling a package
> npm uninstall react-devtools
> pnpm remove react-devtools
```

Figure 6: Example of installing packages with NPM and PNPM

PNPM claims to be as fast as NPM and Yarn, and in some cases to be faster than both in their official benchmarks. (PNPM Official Benchmarks 2019) These tests would be hard to verify as we do not know what conditions these benchmarks were performed under (Network speeds, hardware, etc.)

The current market share of PNPM is only around 0.1% (Node.js User Survey 2018) which may be a result of several factors, such as the development team lacking the resources of NPM Inc. and Facebook have in relation to developer outreach, or simply developers unwilling to switch from their current package manager.

### 3 Survey of JavaScript developers

As a part of this research, a survey was designed and sent out to JavaScript users (utilizing such avenues as school WhatsApp groups, the FreeCodeCamp forums, fellow employees at my workplace and Reddit), aiming to gather important data on the attitudes and thoughts on JavaScript developers towards package managers and related subjects.

The survey was designed with Google Forms, due to its ease of use and automatic graphing of the submitted results, allowing for easy parsing of the data for the purposes of this thesis.

The sample was not randomized, and the sample size was naturally limited by the amount of people that were willing to fill in the survey, thus the results of this survey cannot be used to generalize the wider developer community.

#### 3.1 The given questions

The survey consisted of 19 questions and received responses from 29 people. The given questions are listed below (questions that were not mandatory are marked with an X)

- Are you currently employed in the software development field?
- How long have you been programming in JavaScript for?
- How did you first hear about JavaScript package managers?
- Do you currently use a JavaScript package manager?
- Which of the listed package managers have you heard of?
- Of those listed, which package managers have you used?
- What package manager do you use mainly?
- If you switched from one package manager to another, can you briefly explain what made you switch? (X)
- How important have package managers become to your development workflow?
- How would you rank the learning curve when using package managers?
- If you have experience using package managers for other programming languages or operating systems, how would you rank the learning curve compared to them? (X)
- When do you commonly encounter error messages when using package managers?
- How would you rate the clarity and the helpfulness of the error messages that you receive from your package manager? (X)
- Would you recommend the use of package managers to another student/developer/hobbyist?
- If your package manager supports running security audits on installed packages, how many times have you run the audit command when working on a project?
- Have you ever heard of an open source package hosted by a package manager database being compromised?
- How concerned are you about the potential of a package installed by the package manager being compromised in some way?
- What do you personally think could be improved in JavaScript package managers? (X)

- Do you have any additional thoughts about this survey that were not covered in the above questions? (X)

### 3.2 Analysis of the gathered data

Listed below are the results of the survey, with questions grouped together by topic.

#### 3.2.1 Questions 1-7: Developer background

These questions were designed to gather background on the surveyed developers and their current knowledge of the package managers asked about.

Are you currently employed in the software development field?

39 responses

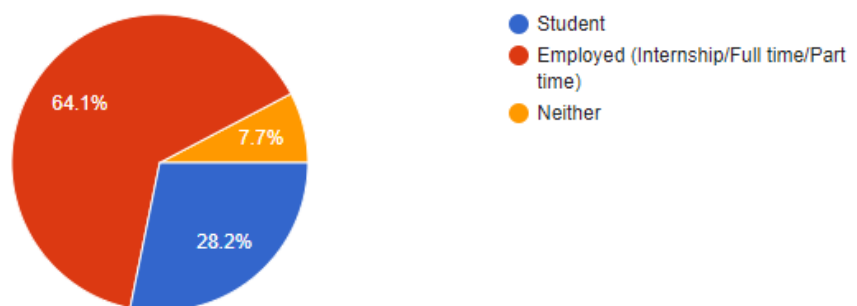


Figure 7: Question 1 survey results

25 people were employed in some form, 11 were students and only 3 were neither (potentially hobbyists, looking for work, etc.)

## How long have you been programming in JavaScript for?

39 responses

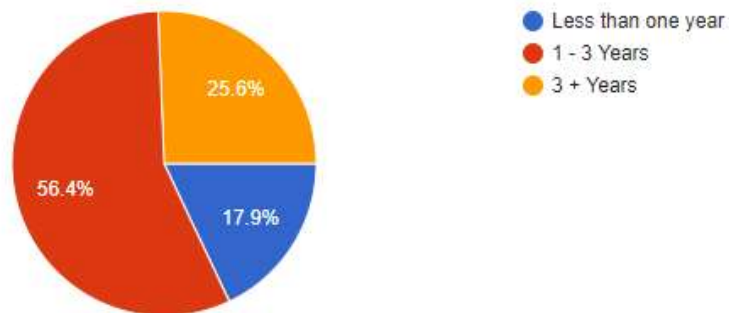


Figure 8: Question 2 survey results

Most respondents (22 people) have programmed in JavaScript for 1-3 years, 10 have been working 3+ years and 7 have less than a year of experience.

## How did you first hear about JavaScript package managers?

39 responses

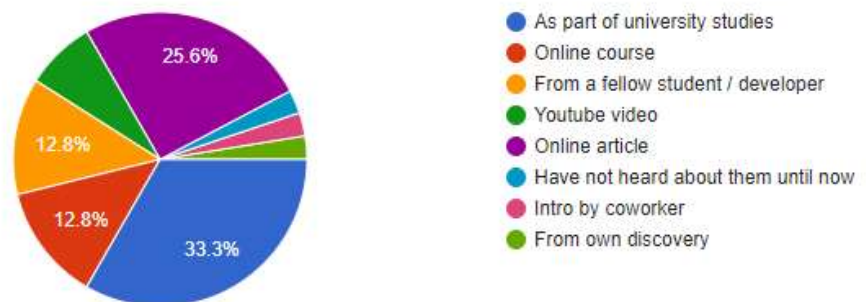


Figure 9: Question 3 survey results

13 of respondents learnt of package managers as part of their university studies, 10 from online articles, 5 from an online course, 5 from fellow student / developers, 3 from YouTube videos, 1 was introduced by a co-worker, 1 discovered them on their own and 1 had not heard about them until asked to respond to this survey.



## Do you currently use a JavaScript package manager?

39 responses

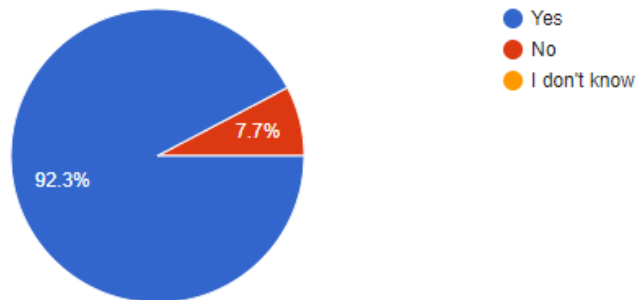


Figure 10: Question 4 survey results

Most respondents (36 people) are currently using package managers and only 3 are currently not using them.

None of the respondents were not sure if they were currently using them in their workflow.

## Which of these package managers have you heard of?

39 responses

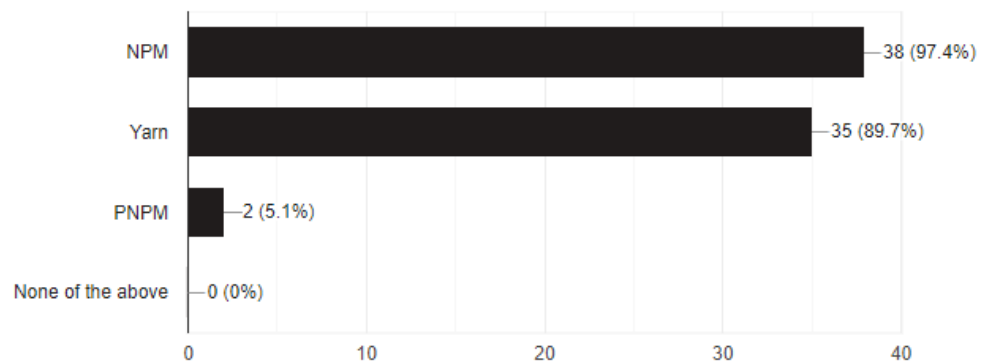


Figure 11: Question 5 survey results

As I personally expected, most respondents had heard of NPM (38 people) and Yarn (35 people), and only 2 had heard of PNPM.

Nobody responding had not heard of any of the package managers listed.

### Of those listed, which package managers have you used?

39 responses

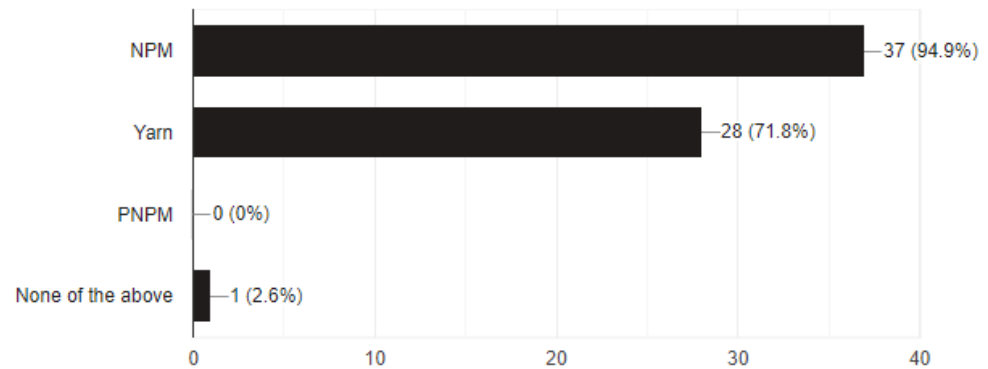


Figure 12: Question 6 survey results

Again, the results of this question fell in line to my expectations: 37 people had used NPM, 28 for Yarn, and none for PNPM. One person was the outlier, having used none of them previously.

### What package manager do you use mainly?

39 responses

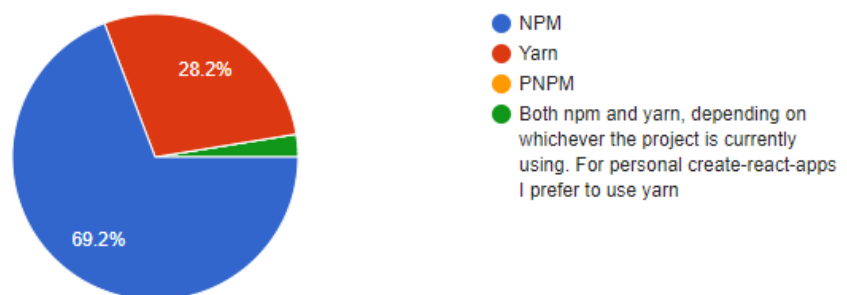


Figure 13: Question 7 survey results

27 people mainly use NPM, 11 for Yarn and 1 person used both NPM and Yarn depending on what the situation called for. None used PNPM mainly.

### 3.2.2 Questions 8-11, 14: Experience with package managers

These questions were designed to gauge how important package managers were to the surveyed developers, how easy they found them to be and if they have switched to another package manager for whatever reason.

For question 8 (“If you switched from one package manager to another, can you briefly explain what made you switch?”) the following responses were submitted from 15 respondents:

- Speed
- A learning curve smaller than the ones I use and innovations
- I did try yarn since facebook suggested it but I just find NPM easier since it comes installed with node hassle free.
- Existing projects
- NPM to Yarn. Yarn seems faster.
- Did not see any bad things in yarn but seemed faster
- npm was really slow/buggy (especially in building packages, such as sass) a year or two ago
- Switched from npm to yarn because it seems to never have the strange issues that npm sometimes has
- Switched from npm to yarn back when yarn was a lot faster, very recently (couple~few months) switched back to npm since apparently npm is pretty equivalent nowadays. Will probably switch back to yarn when yarn PnP matures cos f\*ck node\_modules dawg
- I heard Yarn had faster installation, but my laptop is freaking slow anyways, so I don't notice the difference that much. I use yarn every now and then.

How important have package managers become to your development workflow?

39 responses

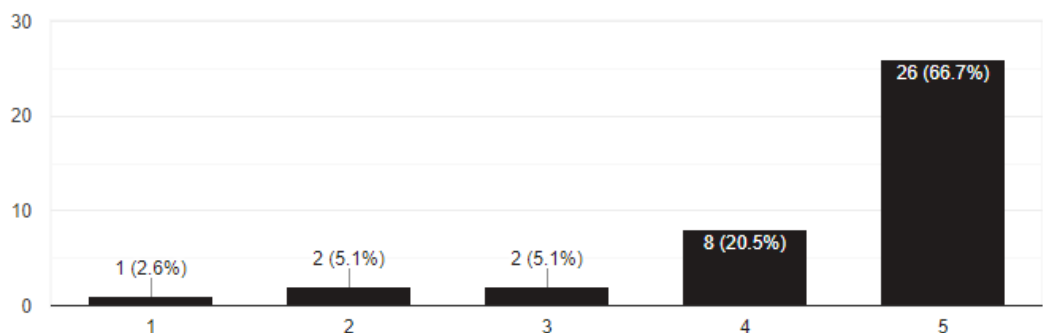


Figure 14: Question 9 survey results

There was not much disagreement for this question, the majority (26) ranked the importance of package managers the highest option possible, 8 ranked it one lower, 2 chose three out of five, another two ranked it as a two and only 1 chose the lowest option.

### How would you rank the learning curve when using package managers?

39 responses

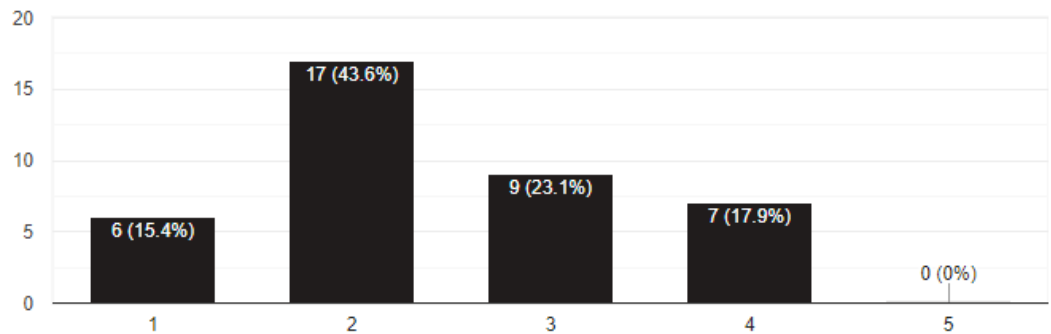


Figure 15: Question 10 survey results

From the easiest (one) to the hardest (five) option, 17 ranked learning to use package managers as a two on the difficulty curve, 6 ranked it as very easy at a one, 9 as slightly harder at a three and 7 as a four. No respondents ranked it as very hard.

### If you have experience using package managers for other programming languages / operating systems, how would you rank the learning curve compared to them?

32 responses

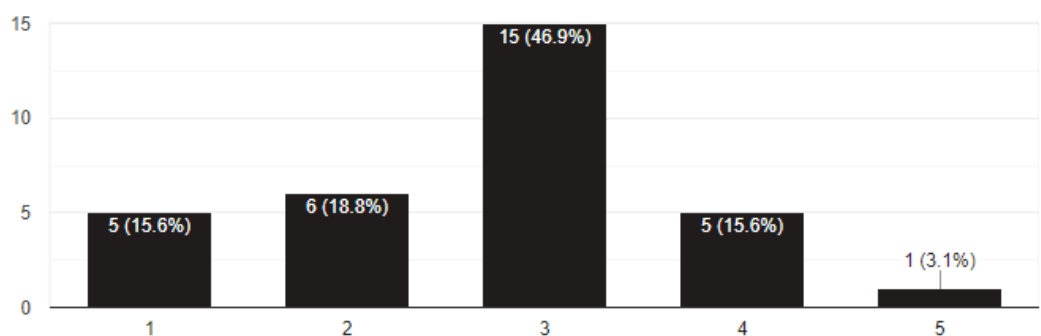


Figure 16: Question 11 survey results

Again, going from lowest (very easy) to highest (very hard) almost half (15 people) chose the difficulty curve compared to other package managers as in the middle at a three, 5 as very easy, 6 as slightly easier at a two, 5 as quite hard at four and only one as very hard.

Would you recommend the use of package managers to another student / developer / hobbyist?

39 responses

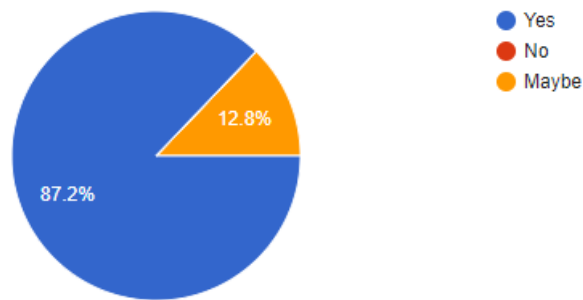


Figure 17: Question 14 survey results

The results for this question are quite definitive, 34 would recommend the use of package managers to a fellow JavaScript developer and 5 might recommend their use. No respondents would recommend against their use.

### 3.2.3 Questions 12-13: Error messages

These questions asked about if the surveyed developers encountered error messages using their package managers and if they found the given messages helpful to them

When do you commonly encounter error messages?

39 responses

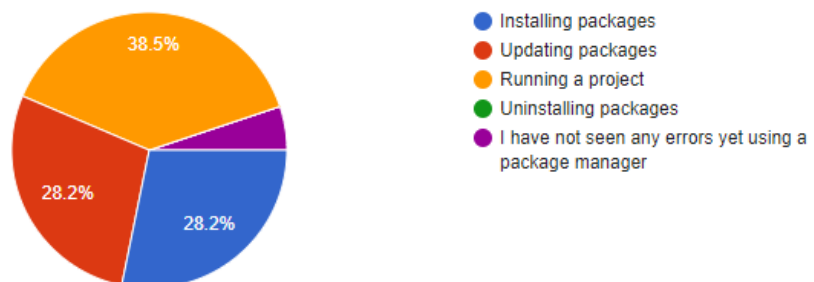


Figure 18: Question 12 survey results

When those surveyed encounter error messages, 15 people commonly encountered them when running a project, 11 when installing packages, 11 when updating packages and only 2 people had not yet encountered any error messages when using their package manager.

How would you rate the clarity and the helpfulness of the error messages that you receive from your package manager?

38 responses

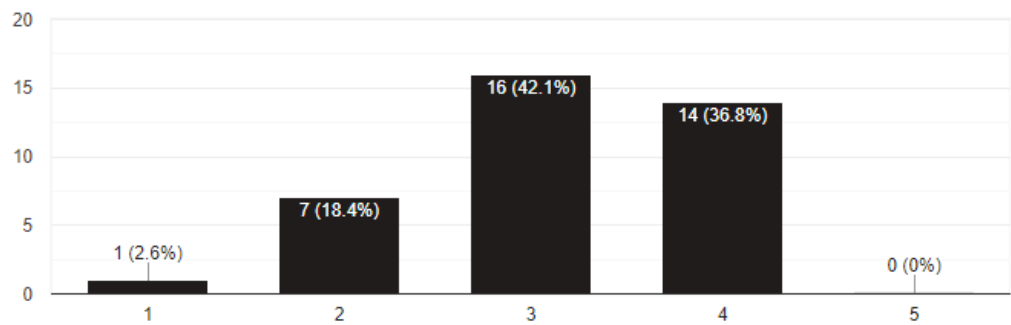


Figure 19: Question 13 survey results

While none selecting the highest option when gauging their satisfaction with the given error messages, most seem generally happy with the clarity of the given error messages with 30 people choosing either a 3 or a 4 to rank their satisfaction. One sole person was unsatisfied with their clarity.

### 3.2.4 Questions 15-17: Security

These questions asked about potential security issues related to package managers and if they have encountered any problems related to security and how much they consider it a potential concern.

If your package manager supports running security audits on installed packages, how many times have you run the audit command when working on a project?

39 responses

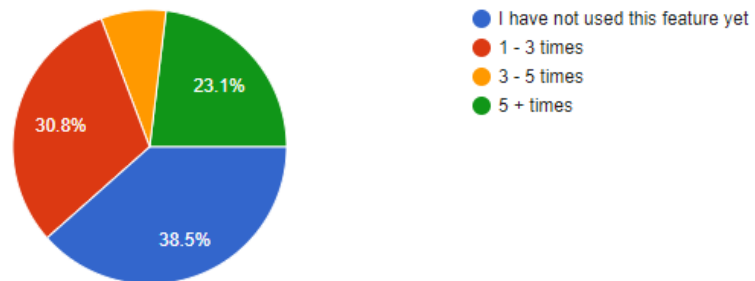


Figure 20: Question 15 survey results

Amongst those surveyed, 15 people said that they have not used the package audit functionality of their package managers. 9 people have used the feature more than 5 times when working on a project, 12 used it 1-3 times and 3 people have used it 3-5 times during a project.

Have you ever heard of an open source package hosted by a package manager database being compromised?

39 responses

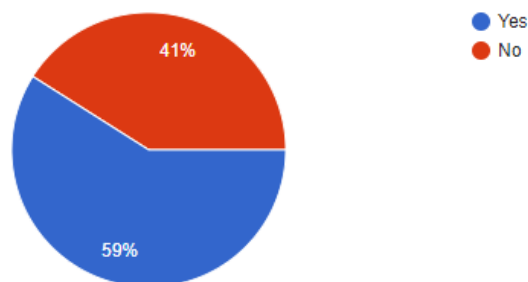


Figure 21: Question 16 survey results

Most people surveyed (23 of 39 people) have heard of a package hosted on a package repository being compromised in some way, while 16 had not heard of it happening.

How concerned are you about the potential of a package installed by the package manager being compromised in some way?

39 responses

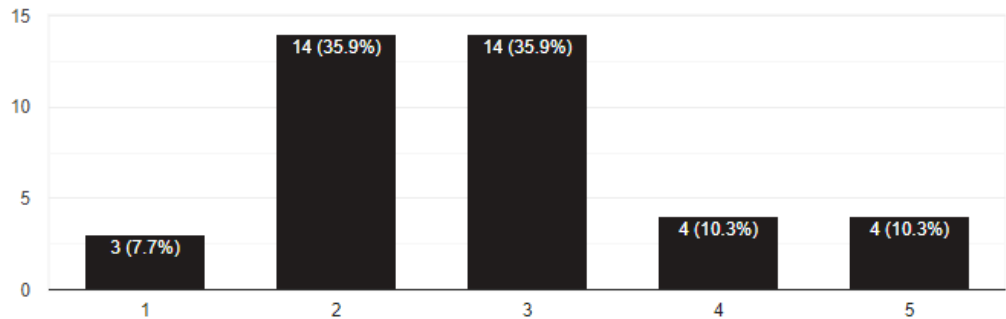


Figure 22: Question 17 survey results

The results of this question seem to indicate the majority of those surveyed have no strong concerns related to possibly installing compromised packages (28 people ranking their concern either a 2 or 3 out of 5) while 3 people ranking their concern at the lowest option. 8 people ranked their concerns either a 4 or 5.

### 3.2.5 Questions 18-19: Additional thoughts

These two questions were optional spaces for developers to leave any additional thoughts that the previous questions may not have allowed them to convey.

For the question “What do you personally think could be improved in JavaScript package managers?” the following comments were left:

- The amount of dependencies is mounting up so we don't really know what all the dependencies are and what they are doing. Because of this I feel there should be more focus on security.
- Maybe have a UI based application to see outdated packages and update them and also see all the packages installed. Something like cakebrew.
- I haven't felt the difference yet
- NPM in particular duplicates packages by not caching versions in a single location, and global installs are a stupid idea
- Download packages faster
- Javascript needs to be centralized in something, being an old language and that evolved quickly in a short time, it ended up becoming a mess. More or less the



same thing happened to php at a time when php ceased to be a scripting language to become a robust and professional language...

- Security (relating to the whole fiasco with event-stream)
- A more intuitive auditing / audit fix process
- package.lock should be added to git(?), but is a mess and rewritten all the time anyway. that could work better
- Should be easier to see the size / effect on bundle size of a package
- What Yarn PnP is doing to reduce duplicate node\_modules and cringeworthy amount of files in total. Error messages - npm's error messages are easy once you know what to ignore but for a beginner jebus lord all mighty.
- Perhaps making it easier to keep your packages secure, though the notifications from github are quite useful already
- It should reduce the size, i.e. stop downloading same library if multiple packages requires it. I think PNPM does that or was it Lerna?

For the question "Do you have any additional thoughts on the subject of this survey that were not covered in the above questions?" the following comments were left (comments that were clearly jokes or one-word answers were removed):

- I've heard the package managers work better depending on the OS?
- What a sh\*tshow this JS ecosystem is, oh my. If only JS had had a standard library by the time it took off.
- I said I encounter errors while installing packages, but for the most part, those are due to typos or using a deprecated way of installing a package

## 4 Hardware Test

To monitor the comparative performance of the three package managers scoped for this thesis, tests were performed over a range of different combinations of hardware and operating systems.

The installation times for packages was measured under a different range of conditions:

- Clean installation
- With the lock file
- With the cache
- With the node modules folder
- A re-installation (lock file, cache and node modules)

### 4.1 Hardware used

The hardware configuration used for these tests is listed below. To ensure consistency of performance, all the operating systems used were installed on a virtual machine using the VirtualBox software onto my desktop computer.

Due to these tests being performed on hardware that I personally own, these results may vary if they were performed on a wider range of hardware configurations or newer hardware.

Despite this, I believe these tests will be indicative of what can be expected when these package managers are used for common developer tasks on similar hardware.

#### 4.1.1 Computer Specifications

- Operating System 1: Windows 10 Education 10.0.17763 Build 17763
- Operating System 2: Ubuntu 19.04
- CPU: Intel i7-9700k
- RAM: 16GB
- Hard drive: 500GB M.2 SSD
- GPU: Nvidia RTX 2070

#### 4.1.2 Operating Systems tested

- Windows 10 Education 10.0.17763 Build 17763
- Ubuntu Linux 19.04
- MacOS 10.11 El Capitan

#### 4.1.3 VirtualBox specifications

- RAM: 4096 MB
- Disk size: 10GB
- Video memory: 128MB
- Processor Count: 4

### 4.2 Test results

To test the installation speeds, the Wiki.js open source project was used (Github Wiki.js 2019). The repository was cloned onto the test machines and the tests were performed using it.

#### 4.2.1 Windows 10

	NPM	Yarn	PNPM
Clean	60.205s	52.80s	43.44s
Lock file	32.956s	54.34s	26.00s
Cache	34.791s	49.10s	24.14s
Node_modules	6.755s	77.97s	3.30s
Reinstall	6.891s	0.55s	5.20s

Figure 23: Windows 10 test results

PNPM comes out ahead of the other package managers with significant differences in installation times compared to the other package managers tested.

#### 4.2.2 MacOS

	NPM	Yarn	PNPM
Clean	110.42s	71.90s	79.33s
Lock file	55.952s	95.06s	89.23s
Cache	48.519s	60.72s	68.58s
Node_modules	9.705s	84.57s	3.98s
Reinstall	11.869s	0.71s	1.97s

Figure 24: MacOS test results

The results of this test were more varied with none of the package managers being consistently faster than the others.

#### 4.2.3 Ubuntu 19.04

	NPM	Yarn	PNPM
Clean	32.375s	41.56s	42.349s
Lock file	22.107s	10.89s	6.41s
Cache	29.266s	38.41s	42.35s
Node_modules	5.611s	40.80s	10.01s
Reinstall	6.056s	0.47s	2.43s

Figure 25: Ubuntu 19.04 test results

NPM performed the fastest in 3 out of the 5 tests performed. The uneven performance of Yarn was quite surprising, as some of the answers given in the previous survey mentioned switching to Yarn based on supposed speed improvements.

## **5 Discussion**

### **5.1 Reflection on my learning during the thesis process**

In my opinion I believe that the thesis process went as smooth as it could have, with no significant problems arising during the thesis work. Only a few minor problems arose, and I managed to complete the thesis project despite them.

Finding time to work on this thesis was a minor struggle as I had to juggle my time between my work as a software developer and the time needed to complete the thesis. Despite this I feel that the fact I was working at the time helped this research, as my experiences using NPM for my JavaScript programming during work gave me valuable working knowledge on it and inspired me to choose this topic for my bachelor's thesis.

During the research I had to rely mainly on online resources as I struggled to find literature resources pertaining to this specific area of research. References to package managers in literature mainly involved books about learning web development that include tutorials for installing and using NPM to develop code, but nothing relating to a comparison of different package managers.

If I was to conduct research on this topic again, I would make sure to find a way to widen the sample size of my survey. Finding respondents for the survey was a struggle, so this would be a main problem area if it were to be conducted again. For the tests, I would need to acquire a wider range of hardware to perform the tests with and expand the scope of the tests to include any new feature sets that may be implemented in the selected package managers. The market share of the package managers may also change, with new ones being introduced and popular ones maybe seeing a reduced market share so this would also need to be considered.

I believe that the skills I have gained in researching and writing for this thesis work will positively benefit me in my future career, especially if I later decide to continue my studies in a master's degree program.

### **5.2 Awareness and importance of package managers**

The first thing that struck me from the gathered data was how vital package managers have become in the world of JavaScript development. Most of those surveyed currently use a package manager.

Out of those, NPM clearly dominated in usage and awareness. Understandable, since it comes bundled with Node.js and for a portion of web developers it will most certainly be their first encounter with any form of package manager. Yarn has the backing of Facebook and trails behind NPM in terms of usage and awareness but significantly leads every other known JavaScript package manager. The two combined have over 60% of market share.

While PNPM clearly lacks the awareness or usage statistics of the others, it can boast generally better performance on Windows 10 than the competitors.

### **5.3 Performance**

A surprising result of this research was that except under certain conditions, none of the package managers can boast to be clearly faster than the others for all use cases.

This was especially surprising to me as when respondents were asked why they had switched to another package if they had done so before, several comments were left saying they had switched to Yarn because of supposed benefits in performance. My contradictory test results seem to indicate that significant work has been done by the NPM team to bring performance to almost parity with Yarn.

### **5.4 Potential areas of improvement**

Several free choice answers were left on the survey when respondents were asked what they would like to see improved in current package managers. The most prominent answers given were focused on the number of packages downloaded and the duplication of them and security.

The problem of duplicated packages and a lack of package caching is something that PNPM was designed specifically to fix, so if this is a user's sole concern relating to package managers, recommending PNPM to these user's would be an easy decision to make and the quickest solution to their problem.

Several comments singled out security as an area in need of improvement. Specifically, that the auditing process was not intuitive enough and it should be easier to keep packages secure. In the next section I will discuss the concern of security in more detail.

Other topics that were mentioned in the free-choice section included suggestions that JavaScript "be centralized in something" and the development of a "standard library" for the language, like what other languages have such as C++ and Java (a standard library is

code that comes with the language that includes additional functionality that needs to be manually imported into your code, such as networking libraries or input/output handling).

The development of a standard library is currently being considered as something to be added into a later version of the ECMAScript standard (TC39 Github 2019). While this suggestion is not directly related to the area of package managers, it is a possibility that if a standard library is implemented into JavaScript containing built-in features that in the past would have been more convenient to implement with third-party libraries, the average amount of packages installed for a project may reduce.

## 5.5 Security

Security has become a significant concern inside and out of the JavaScript community ever since several high-profile incidents of compromised packages making it out into the wild. A recent example would be when the maintainer of the 'event-stream' package handed over control to an interested third party, who then modified the package to inject malicious code designed to steal wallet keys for users of Bitcoin wallets (The New Stack 2018)

As such, this seeming ambivalence is concerning to me, especially considering the fact that based on the results of the survey, most respondents had heard of an incident regarding a compromised package, more than a third had not used the package audit feature of their chosen package manager (if it contained one).

At the time of writing, NPM has the strongest security features: It will inform you of compromised packages after installs, contains an auditing feature to run on your project to detect compromised packages, and the ability to run a command to download new versions of the packages that are not compromised (If fixing the package may lead to breaking other packages, it will not fix it but will allow you to do it if you pass the `--force` flag to the package fixing command). Yarn will also inform you of compromised packages and will let you run an audit on your project but does not currently have the same package fixing feature as NPM (Yarn Github Open Issue 2019). PNPM currently has none of these features.

While the package fixing feature of NPM is a valuable addition, the fact that it is optional may lead some to not bother with it until it is too late, due to a lack of concern or simple forgetfulness. For this, I believe the best course of action would be to make this a mandatory part of the installation process that happens without input from the user (while leaving the option to manually fix packages that may lead to breaking changes in the project).

This could significantly cut down on the number of compromised packages being installed onto developers' machines.

## 5.6 Observations of the ranked questions

For the ranked questions, when the data is grouped together based on the length of time the respondents have worked with JavaScript and the averages of the answers are calculated, some illuminating data is revealed:

	Less than a year	1-3 years	3+ years
Importance of package managers (Higher is better)	3.14	4.77	4.60
Learning curve of package managers (Lower is easier)	2.33	2.35	2.9
Clarity and helpfulness of error messages (Higher is better)	2.57	3.18	3.30
Concern about the potential of compromised packages being installed (Higher is more concerned)	2.57	2.77	3.00

Figure 26: Averages of ranked questions grouped by length of time working with JavaScript

Judging by these results, JavaScript developer attitudes change as they gain more experience in the field. Respondents who have used JavaScript for over a year rank the importance of package managers very high compared to those who have less than a year of experience.

The learning curve ranking only raises 0.02 points from less than a year of experience to 1-3 years of experience but jumps by 0.55 points at 3+ years of experience. Developers with 3 or less years of experience may be using package managers to perform similar

tasks, but the most experienced developers may experience a slightly higher learning curve as they perform more advanced tasks (using scripts, managing different package manger versions for different projects, etc.)

The clarity of the error messages become clearer to those with more experience compared to those with less, although the ranking for those with more experience still only stays around a 3 out of 5. A possible explanation could be that the experienced developers simply understand from prior experience what the error messages mean from trial and error or are now better at researching the meaning of these error messages. Either way, increasing the clarification of the given error messages would be quite valuable for newer developers using this software (A comment was left in the survey alluding to the fact that the error messages could be hard for newer developers to understand)

The concern about the potential of installing a compromised package slightly increases based on the length of experience, but never rises higher than 3 which ties into the seeming ambivalence I've observed when it comes to JavaScript security issues.

## **6 Conclusion**

After conducting my research for this thesis, I cannot strongly recommend the use of one package manager over the other. It is quite simple to install them all to test them for yourself and you may find that one may function better for you based on your operating system and hardware combination.

As all the software tested download their packages from the NPM registry there is no risk that a package will be unavailable to you if you use one over the other. All package managers use a similar set of commands to operate them thus the time investment needed to test them by yourself is very low.

While it seems each of the package managers have done work to improve performance to the point where there is no longer any significant performance benefit using one over the other, there seems to be some room for improvement in certain aspects of the package managers. Based on the results of the survey and opinions left by the respondents, the main areas that could be improved are the increased clarity of given error messages and improvement of auditing features (or making it a default action that is performed on all installations), so that they are easier for end users. Despite these problems, using a package manager for your workflow is still strongly recommended by respondents.



While I am not able to strongly recommend either NPM or Yarn based solely on performance reasons, I would currently recommend the usage of NPM over the others as it has the strongest security features out of all three package managers tested. I cannot recommend using PNPM at this time until it adds package auditing features. even if you have not currently encountered any security issues with using third-party code, the ability to check if any of your packages is a potential security risk is a valuable feature that can help combat the spread of malicious code.

## References

Brendan Eich 2008. Blog post by Brendan Eich. URL:

<https://brendaneich.com/2008/04/popularity/>. Accessed: 6<sup>th</sup> June 2019.

Debian 2017. Description of package managers. URL:

<https://web.archive.org/web/20171017151526/http://aptitude.alieth.debian.org/doc/en/pr01s02.html>. Accessed: 6<sup>th</sup> June 2019.

NPM Documentation 2019. Documentation page about package file. URL:

<https://docs.npmjs.com/files/package.json.html>. Accessed: 6<sup>th</sup> June 2019.

NPM Documentation 2019. Documentation page about creating a package.json file, the example being used for Figure 4. URL: <https://docs.npmjs.com/creating-a-package-json-file>. Accessed: 6<sup>th</sup> June 2019.

NPM Documentation 2019. Documentation page about package-lock file. URL:

<https://docs.npmjs.com/files/package-lock.json>. Accessed: 6<sup>th</sup> June 2019.

Debian 2015. README and source code of earliest version of DPKG package manager for Debian Linux. URL:

<https://web.archive.org/web/20150402141229/https://anonscm.debian.org/cgiit/dpkg/dpkg.git/plain/scripts/perl-dpkg.pl?id=1b80fb16c22db72457d7a456ffbf1f70a8dfc0a5>. Accessed: 6<sup>th</sup> June 2019.

Node.js Github 2019. Repository for Node.js. URL: <https://github.com/nodejs/node-v0.x-archive/tags?after=v0.0.4>. Accessed: 3<sup>rd</sup> April 2019.

NPM Github 2019. Repository for NPM. URL:

<https://github.com/npm/npm/releases?after=v0.1.1>. Accessed: 2<sup>nd</sup> April 2019.

PNPM Github 2019. Repository for PNPM. URL:

<https://github.com/pnpm/pnpm/releases/tag/0.19.0>. Accessed 2<sup>nd</sup> April 2019.

Node.js User Survey 2018. User survey of Node.js users. URL: <https://nodejs.org/en/user-survey-report/>. Accessed: 2<sup>nd</sup> April 2019.

Linux.com 2017. News article about NPM. URL:

<https://www.linux.com/news/event/Nodejs/2016/state-union-npm>. Accessed: 2<sup>nd</sup> April 2019.

NPM Inc. 2019. Services offered by NPM Inc. URL: <https://www.npmjs.com/products>. Accessed: 2<sup>nd</sup> April 2019.

Yarn Github 2019. Repository for Yarn. URL:

<https://github.com/yarnpkg/yarn/releases?after=0.4.0>. Accessed: 3<sup>rd</sup> April 2019.

Code 2019. Announcement of Yarn by Facebook. URL: <https://code.fb.com/web/yarn-a-new-package-manager-for-javascript/>. Accessed: 3<sup>rd</sup> April 2019.

Yarn Homepage 2019. Official Yarn page. URL: <https://yarnpkg.com/en/>. Accessed: 3<sup>rd</sup> April 2019.

PNPM Homepage 2019. Official PNPM page. URL: <https://pnpm.js.org/>. Accessed: 3<sup>rd</sup> April 2019.

PNPM Official Benchmarks 2019. Performance benchmarks of PNPM. URL:

<https://github.com/pnpm/benchmarks-of-javascript-package-managers>. Accessed: 3<sup>rd</sup> April 2019.

MDN 2019. Resources on ECMAScript standards. URL: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language\\_Resources](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources). Accessed: 16<sup>th</sup> April 2019.

Brendan Eich 2007. Blog post by Brendan Eich. URL:

<https://brendaneich.com/2007/10/open-letter-to-chris-wilson/>. Accessed: 16<sup>th</sup> April 2019.

Node.js Docs 2019. Documentation page for Node.js module system. URL:

<https://nodejs.org/dist/latest-v11.x/docs/api/modules.html>. Accessed: 16<sup>th</sup> April 2019.

Github Wiki.js 2019. Repository page for Wiki.js project. URL:

<https://github.com/Requarks/wiki>. Accessed 5<sup>th</sup> May 2019.

The New Stack 2018. News article about event-stream package. URL:

<https://thenewstack.io/attackers-up-their-game-with-latest-npm-package-compromise/>  
Accessed 5<sup>th</sup> May 2019.

TC39 Github 2019. Github page discussing proposal for the inclusion of a standard library into the ECMAScript standard. URL: <https://github.com/tc39/proposal-javascript-standard-library>. Accessed: 15<sup>th</sup> June 2019.

Yarn Github Open Issue 2019. Open issue on Github page discussing lack of fixing audited packages on Yarn. URL: <https://www.github.com/yarnpkg/yarn/issues/7075>.

# Appendices

## Appendix 1. Table of figures

Figure 1: Example of manually importing a code library into a web page without a package manager .....	2
Figure 2: An example of using the ECMAScript 6 module system for importing and exporting code .....	4
Figure 3: An example of using the Node.js module system for importing and exporting code.....	5
Figure 4: Example of what a package file would look like when using NPM, taken from the NPM Documentation (NPM Documentation 2019) .....	6
Figure 5: An example of using different commands with NPM and Yarn .....	8
Figure 6: Example of installing packages with NPM and PNPM.....	9
Figure 7: Question 1 survey results.....	11
Figure 8: Question 2 survey results.....	12
Figure 9: Question 3 survey results.....	12
Figure 10: Question 4 survey results.....	13
Figure 11: Question 5 survey results.....	13
Figure 12: Question 6 survey results.....	14
Figure 13: Question 7 survey results.....	14
Figure 14: Question 9 survey results.....	15
Figure 15: Question 10 survey results.....	16
Figure 16: Question 11 survey results.....	16
Figure 17: Question 14 survey results.....	17
Figure 18: Question 12 survey results.....	18
Figure 19: Question 13 survey results.....	18
Figure 20: Question 15 survey results.....	19
Figure 21: Question 16 survey results.....	19
Figure 22: Question 17 survey results.....	20
Figure 23: Windows 10 test results .....	22
Figure 24: MacOS test results.....	23
Figure 25: Ubuntu 19.04 test results .....	23
Figure 26: Averages of ranked questions grouped by length of time working with JavaScript.....	27